UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Operadores Aritméticos de Baixo Consumo para Arquiteturas de Circuitos DSP

por

EDUARDO ANTONIO CÉSAR DA COSTA

Tese submetida à avaliação
como requisito parcial para a obtenção do grau de
Doutor em Ciência da Computação

Dr. Sergio Bampi
Orientador

Dr. José Carlos Alves Pereira Monteiro
Co-orientador

Porto Alegre, agosto de 2002.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Costa, Eduardo Antonio César da

Operadores Aritméticos de Baixo Consumo para Arquiteturas de Circuitos DSP / por Eduardo Antonio César da Costa. — Porto Alegre: PPGC da UFRGS, 2002.

193 f.: il.

1. Circuitos CMOS. 2. Consumo de potência. 3. Atividade de chaveamento. 4. Correlação temporal. 5. Codificação de sinais. 6. Exploração arquitetural. I. Bampi, Sergio. II. Monteiro, José Carlos Alves Pereira. III. Título.

Agradecimentos

Ao término deste trabalho, quero prestar o meu agradecimento a todas as pessoas que ajudaram direta ou indiretamente nesta trajetória, seja com contribuições técnicas ou com gestos de amizade e companheirismo.

Antes de mais nada, agradeço à Universidade Católica de Pelotas pelo apoio, dando-me a oportunidade de dedicação integral ao doutorado. Também agradeço à Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, pelo apoio, através do Programa de Doutorado Sanduíche.

Aos colegas de mestrado e doutorado do curso de Pós-Graduação em Computação da UFRGS o meu agradecimento pela agradável convivência e troca de conhecimento durante esta jornada. Aos professores do PPGC, com os quais tive a oportunidade de conviver de uma forma mais próxima, agradeço pela contribuição na minha formação profissional. Um especial agradecimento ao professor Luigi Carro pela sua orientação nos meus primeiros passos no PPGC. A sua competente orientação e constante incentivo colaboraram de forma inestimável na minha formação.

Um forte agradecimento aos companheiros do grupo ALGOS do Instituto de Engenharia de Sistemas e Computadores, INESC, em Lisboa, Ana de Jesus, João Portela, José Carlos Costa, Luis Guerra e Silva, Lino Pinchete e João Silva, pelos inúmeros favores prestados e agradáveis momentos compartilhados. Aos professores Arlindo Oliveira e Luis Miguel Silveira agradeço pelas formas gentil e acolhedora com que me receberam no INESC. Um especial e forte agradecimento ao meu orientador no INESC, prof. José Monteiro pela amizade, orientação, apoio, incentivo e pelas inúmeras reuniões técnicas dedicadas ao trabalho, que serviram para fortalecer e consolidar os meus conhecimentos na linha de pesquisa deste trabalho. A todos estes amigos portugueses um grande obrigado.

Aos meus pais Geraldo e Gláucia e a toda a minha família o meu carinho especial e agradecimento pela força e incentivo constantes. À minha esposa Andreia Costa um profundo agradecimento, por ter sido esposa, amiga e companheira em todas as horas. O seu apoio, carinho e compreensão foram as minhas fontes de motivação e inspiração nos momentos necessários.

Finalmente ao meu orientador na UFRGS, prof. Sergio Bampi, um forte agradecimento pela orientação e pela forma acolhedora como me recebeu como seu orientando. Sua amizade, incentivo e apoio foram fundamentais ao longo desta jornada.

Sumário

Lista de Figuras . . . 9

Lista de Tabelas . . . 13

Resumo . . . 15

Abstract . . . 17

1 Introdução . . . 19
1.1 Motivação . . . 20
1.2 Objetivos . . . 21
1.3 Metodologia . . . 22
1.4 Contribuições do Trabalho . . . 23
1.5 Organização do Trabalho . . . 24

2 Consumo de Potência em Circuitos CMOS . . . 27
2.1 Consumo de Potência Estática . . . 27
2.1.1 Exemplos de Consumo de Potência Estática em Circuitos não CMOS 27
2.1.2 Potência Estática Devido às Correntes de Fuga . . . 28
2.1.3 Potência Estática Devido à Corrente de Sublimiar . . . 29
2.2 Consumo de Potência de Curto-Circuito . . . 29
2.3 Consumo de Potência Dinâmica . . . 33
2.3.1 Probabilidade de Chaveamento . . . 34
2.3.2 *Glitching* em Circuitos CMOS Estáticos . . . 35
2.3.3 Componentes de Capacitâncias de Nós . . . 36
2.3.3.1 Capacitância de Porta . . . 36
2.3.3.2 Capacitância de Sobreposição . . . 37
2.3.3.3 Capacitância de Dreno e Fonte . . . 38
2.3.3.4 Capacitância de Conexão . . . 38
2.4 Metodologia de Análise de Circuitos Utilizados neste Trabalho 39
2.4.1 Ambiente de Análise de Potência . . . 40
2.4.2 Análise de Potência Considerando *Glitches* . . . 40
2.4.3 Metodologia de Análises de Área, Atraso e Potência . . . 41
2.5 Resumo . . . 43

3 Técnicas de Otimização de Potência em Circuitos CMOS 45
3.1 Nível de Sistema . . . 46
3.2 Nível de Algoritmo . . . 47
3.2.1 Nível Comportamental . . . 47
3.2.1.1 Técnicas de Escalonamento . . . 47
3.2.1.2 Alocação de Recursos . . . 47
3.2.2 Nível de Instruções . . . 48
3.3 Nível Arquitetural . . . 48
3.4 Nível Lógico . . . 49
3.5 Nível de Circuitos . . . 50
3.6 Nível de Tecnologia . . . 51
3.7 Resumo . . . 51

4 Redução de Potência no Nível Arquitetural 53
4.1 Técnicas de Redução da Atividade de Chaveamento 54
4.1.1 Compartilhamento de Recursos 54
4.1.2 Métodos de Codificação em Barramentos 56
4.1.2.1 Codificação *One-Hot* 57
4.1.2.2 Codificação *Bus-Invert* 57
4.1.2.3 Codificação Gray 58
4.1.2.4 Codificação *Beach* 59
4.1.2.5 Codificação *Transition* 60
4.1.2.6 Codificação *Bit Prediction* 62
4.1.2.7 Codificação *Limited-Weight* 63
4.1.2.8 Complemento de 2 versus Sinal-Magnitude 64
4.1.3 Ordenamento dos Sinais de Entrada 64
4.1.4 Redução da Atividade de *Glitching* 65
4.2 Redução de Potência Utilizando Transformações 66
4.2.1 Transformações em Velocidade 67
4.2.1.1 *Retiming* 67
4.2.1.2 Processamento Paralelo e *Pipelining* 68
4.2.1.3 Aplicação de *Loop Unrolling* 70
4.2.2 Transformações Genéricas 72
4.2.2.1 Redução do Número de Operações 72
4.2.2.2 Substituição de Operações 74
4.2.2.3 Redução do Comprimento da Palavra 75
4.3 Resumo 75

5 Técnicas Desenvolvidas para a Redução de Po-tência . 77
5.1 Codificação para Baixa Potência 77
5.1.1 Técnica *Transition Coding* 78
5.1.2 Código Redundante 79
5.1.3 Código Gray 80
5.1.4 Código Híbrido 82
5.2 Operadores Aritméticos em Código Híbrido sem Sinal 84
5.2.1 Elementos Aritméticos Básicos 84
5.2.1.1 Elemento Básico Somador Híbrido com m=2 84
5.2.1.2 Elemento Básico Multiplicador Híbrido com m=2 85
5.2.2 Multiplicador Híbrido tipo *Array* 86
5.2.2.1 Multiplicador *Array* Binário 86
5.2.2.2 Operação do Multiplicador Híbrido com m=2 88
5.2.2.3 Arquitetura do Multiplicador Híbrido com m=2 89
5.2.2.4 Algoritmo para Geração dos Multiplicadores Híbridos com m=2 . . . 90
5.2.3 Comparação de Desempenho dos Circuitos Multiplicadores *Array* . . . 91
5.2.3.1 Resultados de Área 91
5.2.3.2 Resultados de Atraso 92
5.2.3.3 Resultados de Consumo de Potência 92
5.2.4 Arquiteturas com Operações Aritméticas na Base 2^m 95
5.2.4.1 Elementos Somadores e Multiplicadores Básicos 96
5.2.4.2 Multiplicação na Base 2^m 99

5.2.4.3 Arquitetura do Multiplicador *Array* na Base 2^m . . . 101
5.2.4.4 Comparações de Desempenho . . . 102
5.2.4.5 Carga Capacitiva no Barramento de Dados . . . 104
5.3 Ordenação e Particionamento de Coeficientes . . . 105
5.3.1 Manipulação de Coeficientes em Algoritmos de Filtros FIR . . . 105
5.3.1.1 Resultados da Aplicação da Manipulação de Coeficientes . . . 106
5.3.2 Manipulação de Coeficientes em Algoritmo de Transformada Rápi-da de Fourier . . . 109
5.3.2.1 Resultados da Aplicação da Manipulação de Coeficientes . . . 111
5.4 Resumo . . . 113

6 Arquiteturas de Multiplicadores em Complemento de 2 . . . 115
6.1 Multiplicadores *Array* em Complemento de 2 . . . 115
6.1.1 Trabalho Relacionado . . . 115
6.1.2 Multiplicador na Base 2^m . . . 117
6.1.2.1 Multiplicação Binária em Complemento de 2 . . . 117
6.1.2.2 Multiplicação Binária na Base 2^m em Complemento de 2 . . . 118
6.1.2.3 Multiplicação Híbrida na Base 2^m em Complemento de 2 . . . 120
6.1.2.4 Arquitetura de Multiplicador na Base 2^m em Complemento de 2 . . . 120
6.1.3 Multiplicador Booth Modificado . . . 122
6.1.4 Comparações de Desempenho . . . 124
6.1.4.1 Multiplicadores *Array* com Sinal Usando Diferentes Bases . . . 125
6.1.4.2 Multiplicadores *Array* em Complemento de 2 na Base 4 . . . 127
6.1.5 Arquiteturas na Forma *Pipelined* . . . 129
6.1.5.1 Trabalho Relacionado . . . 129
6.1.5.2 Arquitetura *Array* Base 2^m na Forma *Pipelined* . . . 130
6.1.5.3 Arquitetura Booth Modificada na Forma *Pipelined* . . . 131
6.1.5.4 Comparações de Desempenho . . . 132
6.2 Multiplicação Desloca-Soma em Complemento de 2 . . . 133
6.2.1 Trabalho Relacionado . . . 133
6.2.2 Operação Desloca-Soma em Código Híbrido . . . 134
6.3 Resumo . . . 136

7 Exploração Arquitetural com Técnicas de Baixa Potência: Estudo de Casos . . . 139
7.1 Implementações de Filtro FIR . . . 139
7.1.1 Arquiteturas Totalmente Paralelas . . . 141
7.1.2 Arquiteturas Totalmente Seqüenciais . . . 143
7.1.3 Arquiteturas Semi-Paralelas . . . 145
7.1.4 Comparações das Arquiteturas . . . 147
7.1.5 Aplicação de Operadores Aritméticos na Base 2^m . . . 148
7.1.5.1 Análise de Resultados de Área . . . 148
7.1.5.2 Análise de Resultados de Período Mínimo de Relógio . . . 150
7.1.5.3 Análise de Resultados de Potência por Amostra . . . 150
7.1.6 Comparação de Arquiteturas com Multiplicador Booth . . . 152
7.2 Implementações da Transformada Rápida de Fourier . . . 153
7.2.1 Arquiteturas Seqüenciais . . . 155
7.2.2 Arquiteturas Semi-Paralelas . . . 157

7.2.3 Comparações das Arquiteturas . . . 159
7.2.4 Arquiteturas com Multiplicadores na Base 2^m . . . 160
7.2.4.1 Resultados de Área, Período Mínimo de Relógio e Potência por Transformada . . . 161
7.2.5 Comparação das Arquiteturas com Multiplicador Booth Modificado . 163
7.2.6 Arquitetura FFT com Maior Número de Pontos . . . 164
7.2.6.1 Características da Arquitetura . . . 164
7.2.6.2 Aplicação de Multiplicadores *Array* na Base 2^m . . . 165
7.2.6.3 Comparação com Arquitetura Utilizando Multiplicador Booth . . . 166
7.3 Resumo . . . 167

8 Conclusão . . . 169
8.1 Principais contribuições . . . 171
8.2 Trabalhos Futuros . . . 173
8.2.1 Implementação dos Multiplicadores *Array* no Nível Físico . . . 173
8.2.2 Exploração da Capacitância de Carga . . . 173
8.2.3 Aplicação de Circuitos Somadores Eficientes . . . 174
8.2.4 Manipulação de Coeficientes . . . 174
8.2.5 Aplicação dos Operadores Aritméticos *Array* em Arquiteturas mais Complexas . . . 175

Bibliografia . . . 177

Lista de Figuras

FIGURA 2.1 – Dissipação de potência estática em um inversor CMOS controlado por uma porta de passagem NMOS. 28
FIGURA 2.2 – Dissipação de potência estática em um inversor pseudo-NMOS. 28
FIGURA 2.3 – Representação da corrente de fuga reversa. 29
FIGURA 2.4 – Caracterização da corrente de curto-circuito em um inversor sem carga na saída. 30
FIGURA 2.5 – Gráfico da potência de curto normalizada versus rampa de entrada logarítmica. 32
FIGURA 2.6 – Caracterização da potência dinâmica de uma porta CMOS. . . 33
FIGURA 2.7 – Gráfico da variação das correntes dinâmicas com capacitâncias de carga e rampas. 35
FIGURA 2.8 – Esquema de contribuição de *glitches* para dissipação de potência. 35
FIGURA 2.9 – Modelo das capacitâncias parasitas de uma porta CMOS. . . . 36
FIGURA 2.10 – Capacitância de sobreposição para um componente MOS. . . 37
FIGURA 2.11 – Exemplo de aplicação da técnica simbólica para um modelo de atraso zero. 41
FIGURA 2.12 – Metodologia para estimativa de área, atraso e potência dos circuitos. 42

FIGURA 3.1 – Técnicas de otimização de potência para cada nível de abstração no fluxo de projeto. 46

FIGURA 4.1 – Ilustração das primitivas do nível arquitetural. 53
FIGURA 4.2 – Exemplo de estruturas de compartilhamento de barramentos. . 55
FIGURA 4.3 – Esquemas de transmissão e recepção de dados em barramento. 57
FIGURA 4.4 – Exemplo de aplicação do código *bus-invert*. 58
FIGURA 4.5 – Exemplo de aplicação do código gray. 59
FIGURA 4.6 – Exemplo de aplicação do código *beach*. 60
FIGURA 4.7 – Exemplo de aplicação do código *transition*. 61
FIGURA 4.8 – Estrutura para implementação da técnica de codificação *transition*. 61
FIGURA 4.9 – Estrutura para implementação da técnica de codificação *bit prediction*. 62
FIGURA 4.10 – Redução da atividade de chaveamento pela reordenação das entradas. 65
FIGURA 4.11 – Representação de operações para redução da atividade de *glitching*. 66
FIGURA 4.12 – Filtro IIR de segunda ordem. 67
FIGURA 4.13 – Filtro IIR de segunda ordem após aplicação de *retimings*. . . 67
FIGURA 4.14 – Arquitetura paralela para o circuito somador-comparador exemplo. 68
FIGURA 4.15 – Arquitetura *pipeline* para o circuito somador-comparador exemplo. 69
FIGURA 4.16 – Aplicação de *loop unrolling* na estrutura de um filtro IIR de primeira ordem. 71

FIGURA 4.17 – Aplicação de transformação algébrica e propagação de constantes ao filtro IIR de primeira ordem. 71
FIGURA 4.18 – Aplicação de *pipelining* ao filtro IIR de primeira ordem. . . . 72
FIGURA 4.19 – Redução da capacitância chaveada com manutenção do desempenho. 73
FIGURA 4.20 – Redução de capacitância com aumento do caminho crítico. . 73
FIGURA 4.21 – Substituição de operação de multiplicação por adição. 74

FIGURA 5.1 – Conversão entre os códigos Binário e Híbrido. 84
FIGURA 5.2 – Circuito somador Híbrido de 2 bits. 85
FIGURA 5.3 – Circuito multiplicador Híbrido de 2 bits. 86
FIGURA 5.4 – Multiplicador *array* Binário de 4 bits. 87
FIGURA 5.5 – Algoritmo para a geração de diferentes multiplicadores Binários. 88
FIGURA 5.6 – Operação de multiplicação de 4 bits em código Híbrido. 88
FIGURA 5.7 – Arquitetura do multiplicador Híbrido *array* de 4 bits. 89
FIGURA 5.8 – Algoritmo para geração de multiplicadores Híbridos de diferentes comprimentos de palavra. 90
FIGURA 5.9 – Variação da potência com o número de pontos para multiplicadores de 16 bits. 96
FIGURA 5.10 – Exemplo de um somador Híbrido de 8 bits com m=2. 97
FIGURA 5.11 – Exemplo de uma multiplicação de 8 bits em código Binário na base 16. 100
FIGURA 5.12 – Exemplo de uma multiplicação de 8 bits em código Híbrido na base 16. 100
FIGURA 5.13 – Arquitetura de um multiplicador *array* de 8 bits na base 16. . 101
FIGURA 5.14 – Algoritmo para a geração de particionamento e ordenação de coeficientes. 106
FIGURA 5.15 – Fluxo de dados de uma FFT de fator comum na base 2 com decimação em freqüência de 16 pontos. 108
FIGURA 5.16 – Diagrama de uma borboleta com decimação em freqüência de fator comum na base 2. 109

FIGURA 6.1 – Exemplo de uma multiplicação Binária com sinal em operandos $W = 4$ bits. 118
FIGURA 6.2 – Exemplo de um multiplicador *array* Binário com sinal para operandos $W = 4$ bits. 119
FIGURA 6.3 – Exemplo de multiplicação Binária de 8 bits base 16 em complemento de 2. 120
FIGURA 6.4 – Exemplo de multiplicação Híbrida de 8 bits base 16 em complemento de 2. 122
FIGURA 6.5 – Estrutura geral de um multiplicador base 2^m em complemento de 2. 122
FIGURA 6.6 – Exemplo de multiplicador *array* Binário de 8 bits na base 16 em complemento de 2. 123
FIGURA 6.7 – Exemplo de multiplicador *array* Híbrido de 8 bits na base 16 em complemento de 2. 124
FIGURA 6.8 – Exemplo de multiplicação de 8 bits usando o algoritmo Booth Modificado. 125
FIGURA 6.9 – Exemplo de arquitetura Booth Modificada de 8 bits. 126

FIGURA 6.10 – Exemplo de um multiplicador *array* de 8 bits base 4 na forma *pipelined*. 130
FIGURA 6.11 – Exemplo de arquitetura Booth Modificada de 8 bits na forma *pipelined*. 131
FIGURA 6.12 – Estrutura da operação de multiplicação desloca-soma em código Híbrido. 134
FIGURA 6.13 – Exemplos de multiplicação desloca-soma em código Híbrido com m=2 e 4. 135
FIGURA 6.14 – Arquiteturas dos multiplicadores desloca-soma Híbridos com m=2 e 4. 136

FIGURA 7.1 – Parte operativa das implementações de filtro FIR totalmente paralelas. 141
FIGURA 7.2 – Parte operativa das implementações seqüenciais do filtro FIR. 143
FIGURA 7.3 – Parte operativa das implementações semi-paralelas do filtro FIR. 145
FIGURA 7.4 – Estrutura dos operadores aritméticos para a borboleta utilizada no algoritmo FFT de fator comum com decimação em freqüência na base 2. 154
FIGURA 7.5 – Parte operativa das implementações da FFT seqüenciais. . . . 155
FIGURA 7.6 – Parte operativa das implementações da FFT semi-paralelas. . 158

Lista de Tabelas

TABELA 2.1 – Parâmetros de transistores na tecnologia 0,8μm 38
TABELA 2.2 – Parâmetros físicos para a tecnologia 0,8μm. 39
TABELA 2.3 – Componentes de capacitância de carga para um inversor . 39

TABELA 4.1 – Exemplo de aplicação da técnica LWC. 63
TABELA 4.2 – Resultados de arquiteturas com escalonamento de V_{dd}. . . 70

TABELA 5.1 – Somador completo na representação *Transition Coding*. . . 79
TABELA 5.2 – Exemplo de código Redundante. 79
TABELA 5.3 – Resultados em somadores em códigos Binário e Redundante 80
TABELA 5.4 – Área para operadores em códigos Binário e Gray 81
TABELA 5.5 – Potência para operadores em códigos Binário e Gray . . . 81
TABELA 5.6 – Potência para somadores e multiplicadores de 4 bits 82
TABELA 5.7 – Representações dos códigos Binário, Híbrido (m=2) e Gray. 83
TABELA 5.8 – Transições para os códigos Binário, Híbrido (m=2) e Gray 83
TABELA 5.9 – Área e potência para somadores Binário e Híbrido de 2 bits. 84
TABELA 5.10 – Parâmetros para somador Binário e novo somador Híbrido 85
TABELA 5.11 – Área e potência para multiplicadores Binário e Híbrido . 86
TABELA 5.12 – Área, em literais, para os multiplicadores *array* 91
TABELA 5.13 – Atrasos para multiplicadores *array* Binário e Híbrido. . . 92
TABELA 5.14 – Potência dos multiplicadores *array* (atraso zero) 93
TABELA 5.15 – Potência dos multiplicadores *array* (atraso genérico) . . . 94
TABELA 5.16 – Potência dos multiplicadores *array* (entradas uniformes) . 94
TABELA 5.17 – Profundidade lógica para os multiplicadores *array* 95
TABELA 5.18 – Área e atraso para somadores básicos Binário e Híbrido . 98
TABELA 5.19 – Potência para somadores básicos Binário e Híbrido 98
TABELA 5.20 – Área e atraso para multiplicadores básicos Binário e Híbrido 99
TABELA 5.21 – Potência para multiplicadores básicos Binário e Híbrido . 99
TABELA 5.22 – Área para multiplicadores Binário e Híbrido para W=16 . 102
TABELA 5.23 – Atraso para multiplicadores Binário e Híbrido para W=16 103
TABELA 5.24 – Potência para multiplicadores *array* (entradas *trace* real) . 104
TABELA 5.25 – Potência para multiplicadores *array* (entadas randômicas) 104
TABELA 5.26 – Potência para diferentes cargas capacitivas 104
TABELA 5.27 – Potência de filtro FIR com ordenação de coeficientes . . . 107
TABELA 5.28 – Filtros FIR com ordenação e particionamento de coeficientes 107
TABELA 5.29 – Potência com manipulação de coeficientes e redução de V_{dd} 108
TABELA 5.30 – Ordenação de coeficientes no algoritmo FFT. 110
TABELA 5.31 – Ordenação dos coeficientes particionados no algoritmo FFT. 111
TABELA 5.32 – Arquiteturas seqüenciais (ordenação de coeficientes) . . . 111
TABELA 5.33 – Arquiteturas semi-paralelas (coeficientes particionados) . 112
TABELA 5.34 – Circuitos com manipulação de coeficientes e redução de V_{dd} 112

TABELA 6.1 – Representações dos códigos em complemento de 2 121
TABELA 6.2 – Regras de operação do multiplicador Booth Modificado. . . 123
TABELA 6.3 – Área para multiplicadores *array* em complemento de 2 . . 125
TABELA 6.4 – Atraso para multiplicadores *array* em complemento de 2 . 126

TABELA 6.5 – Multiplicadores em complemento de 2 (entradas *trace* real) 127
TABELA 6.6 – Multiplicadores em complemento de 2 (entradas randômicas)127
TABELA 6.7 – Parâmetros para multiplicadores paralelos de 16 bits . . . 128
TABELA 6.8 – Potência para multiplicadores paralelos com sinais randômicos129
TABELA 6.9 – Multiplicadores em complemento de 2 na forma *pipelined* . 132
TABELA 6.10 – Valores de potência para sinal de padrão randômico. . . . 133

TABELA 7.1 – Parâmetros para arquiteturas de filtro FIR paralelas 142
TABELA 7.2 – Parâmetros para arquiteturas de filtro FIR seqüenciais . . 144
TABELA 7.3 – Parâmetros para arquiteturas de filtro FIR semi-paralelas . 146
TABELA 7.4 – Comparação de potência normalizada por amostra 148
TABELA 7.5 – Área para alternativas arquiteturais na forma *pipelined* . . 149
TABELA 7.6 – Atraso para alternativas arquiteturais na forma *pipelined* . 151
TABELA 7.7 – Potência normalizada para arquiteturas *pipelined* 151
TABELA 7.8 – Potência normalizada com multiplicadores na base 4 152
TABELA 7.9 – Parâmetros para arquiteturas FFT seqüenciais 157
TABELA 7.10 – Parâmetros para arquiteturas FFT semi-paralelas 159
TABELA 7.11 – Potência das arquiteturas FFT com redução de V_{dd} 159
TABELA 7.12 – Área para as arquiteturas e valores de m na forma *pipelined*161
TABELA 7.13 – Atraso para arquiteturas e valores de m na forma *pipelined*162
TABELA 7.14 – Potência nas arquiteturas e valores de m na forma *pipelined*163
TABELA 7.15 – Potência nas arquiteturas com multiplicadores na base 4 . 163
TABELA 7.16 – Potência normalizada por transformada nos circuitos FFT 164
TABELA 7.17 – Potência nas arquiteturas seqüenciais (forma direta) . . . 165
TABELA 7.18 – Potência nas arquiteturas seqüenciais (forma *pipelined*) . 166
TABELA 7.19 – Potência com multiplicadores *array* e Booth (base 4) . . . 167
TABELA 7.20 – Potência com multiplicadores na base 4 na forma *pipelined*167

Resumo

Este trabalho tem como foco a aplicação de técnicas de otimização de potência no alto nível de abstração para circuitos CMOS, e em particular no nível arquitetural e de transferência de registradores (*Register Transfer Level - RTL*). Diferentes arquiteturas para projetos específicos de algoritmos de filtros FIR e transformada rápida de Fourier (FFT) são implementadas e comparadas. O objetivo é estabelecer uma metodologia de projeto para baixa potência neste nível de abstração. As técnicas de redução de potência abordadas têm por objetivo a redução da atividade de chaveamento através das técnicas de exploração arquitetural e codificação de dados. Um dos métodos de baixa potência que tem sido largamente utilizado é a codificação de dados para a redução da atividade de chaveamento em barramentos. Em nosso trabalho, é investigado o processo de codificação dos sinais para a obtenção de módulos aritméticos eficientes em termos de potência que operam diretamente com esses códigos. O objetivo não consiste somente na redução da atividade de chaveamento nos barramentos de dados, mas também a minimização da complexidade da lógica combinacional dos módulos. Nos algoritmos de filtros FIR e FFT, a representação dos números em complemento de 2 é a forma mais utilizada para codificação de operandos com sinal. Neste trabalho, apresenta-se uma nova arquitetura para operações com sinal que mantém a mesma regularidade de um multiplicador *array* convencional. Essa arquitetura pode operar com números na base 2^m, o que permite a redução do número de linhas de produtos parciais, tendo-se desta forma, ganhos significativos em desempenho e redução de potência. A estratégia proposta apresenta resultados significativamente melhores em relação ao estado da arte. A flexibilidade da arquitetura proposta permite a construção de multiplicadores com diferentes valores de m. Dada a natureza dos algoritmos de filtro FIR e FFT, que envolvem o produto de dados por apropriados coeficientes, procura-se explorar o ordenamento ótimo destes coeficientes no sentido de minimizar o consumo de potência das arquiteturas implementadas.

Palavras-chave: Circuitos CMOS, consumo de potência, atividade de chaveamento, correlação temporal, codificação de sinais, exploração arquitetural.

TITLE: "LOW POWER ARITHMETIC OPERATORS APPLIED TO DSP ARCHITECTURES"

Abstract

The main goal of this work is to investigate power optimization techniques for CMOS circuits described at a high level of abstraction, emphasizing the architectural and register transfer (RT) levels. Different specific architectures of FIR filter and FFT algorithms are implemented and compared. The main goal is to establish a methodology which can be applied for power reduction at this level of abstraction. The power optimization techniques target the reduction of the switching activity through architectural exploration and data encoding techniques. One of the techniques investigated in this work uses signal encoding. Power efficient arithmetic modules that operate directly with different codes were developed. The objective is twofold. First, to investigate operand codes that yield simpler, *i.e.*, power efficient, arithmetic modules. Second, to investigate signal encodings that lead to the reduction of the switching activity in the data buses. In FIR filter and FFT algorithms, 2's complement is the most used encoding for signed operands. Thus, we present a new architecture for signed multiplication. The proposed architecture maintains the pure form of an array multiplier. This architecture is extended for radix-2^m encoding, which leads to a reduction of the number of partial lines, enabling large gains in performance and power consumption. The proposed approach significantly improves the state of the art. The flexibility of our architecture allows for the easy construction of multipliers for different values of m. Due to the characteristics of the FIR filter and FFT algorithms, which is performed by the product of input data with appropriate coefficients, the best ordering of these coefficients in order to minimize the power consumption of the implemented architectures is also investigated in this work.

Keywords: CMOS circuits, power consumption, switching activity, temporal correlation, signal encoding, architectural exploration.

1 Introdução

No início da década passada, o consumo de potência em circuitos integrados começou a merecer uma atenção especial. Este aspecto deveu-se principalmente ao surgimento de microprocessadores de alto desempenho que apresentavam consumo de potência da ordem de dezenas de watt [CHA 95]. Com o desenvolvimento dos circuitos integrados, as diversas aplicações tiveram um rápido crescimento, em particular os equipamentos portáteis operados por baterias. Para esses equipamentos, houve uma pressão natural para a redução dos seus consumos de potência, de forma a permitir um aumento do tempo de vida útil das suas baterias. Esses fatores levaram rapidamente a uma demanda por projetos de circuitos integrados de baixa potência. A crescente demanda por equipamentos portáteis tem incrementado estudos cada vez mais avançados na área de economia de potência. Este aspecto está diretamente relacionado ao fato de não haver incremento significativo nos últimos anos da eficiência na tecnologia de baterias [CHA 95, LAN 97, WU 98, BHA 2000, SHA 2002].

Nos últimos anos, o consumo de potência vem sendo apontado como um dos principais parâmetros em projetos de circuitos integrados. Na verdade, o maior desafio diz respeito ao projeto de uma nova geração de produtos que consumam o mínimo de potência, sem comprometer o alto desempenho desejado e sem penalidade excessiva em área de silício. Neste contexto, aspectos de baixa dissipação de potência exigem a pesquisa de novas arquiteturas confiáveis que possam ser testadas e fabricadas e que levem em consideração as necessidades de sistemas eletrônicos portáteis de funções diversas.

O outro fator de interesse por projeto de circuitos considerando a dissipação de potência são problemas de temperatura dos dispositivos. Com a redução de tamanho dos dispositivos, a densidade de projeto tende a aumentar, gerando o aumento do número de componentes dissipando potência por unidade de área. Além disto, a redução de tamanho provoca a redução do atraso de propagação, permitindo a operação dos circuitos em freqüências mais altas, o que por sua vez leva a um aumento do consumo de potência.

Na verdade, a ênfase crescente no projeto de circuitos integrados de baixa potência deve-se também à redução de potência em sistemas de alto desempenho que levem em consideração aspectos como densidade de integração, velocidade de operação e freqüência de relógio.

Este desafio passa necessariamente por questões relacionadas com a quantidade de transistores que se pode utilizar em um sistema digital. Segundo [ITR 99], no ano de 2011 já será possível a integração de 1 bilhão de transistores em um único circuito integrado. Estas projeções devem levar em consideração a análise de limites teóricos e práticos que possam definir o desempenho de um circuito integrado, além de parâmetros físicos, tais como as interconexões [HAM 99, BHA 2000, JEF 2001, SYL 2001].

Para a análise dos limites teóricos e práticos, alguns aspectos passam a ser fundamentais na avaliação de potência, tais como a avaliação da parte dominante do consumo de potência, a estimativa da potência total ou o conhecimento da potência consumida em diferentes partes do circuito integrado.

Com o conhecimento dos diferentes aspectos que envolvem o consumo de potência de um circuito integrado, pode-se explorar o potencial de aplicação de

diferentes técnicas de projeto para a redução de potência.

A redução da tensão de alimentação a limites mínimos confiáveis tem sido historicamente o caminho mais efetivo para redução da dissipação de potência em circuitos lógicos e circuitos de memória [BHA 2000]. De acordo com as projeções em [NTR 97], a tensão de alimentação será reduzida para valores próximos a 510mV no ano de 2012 para uma geração de circuitos CMOS de 50nm e 10GHz de frequência de relógio. Neste caso, o principal desafio no projeto de circuitos de baixa potência será manter o desempenho para tensões e freqüências de operação nesta ordem de grandeza, levando em consideração os requisitos de alto desempenho [BHA 2000].

1.1 Motivação

Tem sido desenvolvida pesquisa intensiva em métodos de projeto para baixa potência considerando os diferentes componentes de um circuito [CHA 95, DEV 95, RAB 96, DEV 96]. Pesquisas iniciais se direcionaram para os níveis de circuitos e portas lógicas [ATH 94, ALI 94, BUR 95, MON 95, BEN 96, MON 96, MON 96a, CON 97, GON 97, MON 99]. A tendência nos dias de hoje é a investigação de técnicas de baixa potência em níveis mais altos de abstração de projeto (arquitetural, algorítmico e sistemas) [CHA 92, LAN 94, MAR 94, CHA 95a, MAR 95, KUM 95, MEH 96a, ROY 96, LEE 97, KOM 98, BEN 99, NAC 2000, TAE 2000, BEN 2001], pois é reconhecido que o potencial para redução de potência é muito maior nestes níveis. Este trabalho de doutorado segue esta tendência, com o particular interesse em projetos de arquiteturas de circuitos com reduzida dissipação de potência.

O projeto de um sistema digital complexo no nível arquitetural inicia com sua especificação no nível de algoritmo. Tal especificação é então automaticamente sintetizada para obtenção dos módulos funcionais do circuito. Desta forma, dependendo da classe de algoritmos, a liberdade do projetista pode estar, por exemplo, na paralelização ou seqüencialização das operações do circuito.

Uma classe de algoritmos que tem merecido especial atenção, com aspectos de baixa potência, são os de processamento digital de sinais (*Digital Signal Processing - DSP*). De fato, este interesse tem sido intensificado desde a proliferação de equipamentos eletrônicos portáteis de alto desempenho operados por baterias, tais como telefones celulares, leitores de CD's, equipamentos biomédicos, etc.

Em aplicações DSP, uma das operações mais básicas inclui a filtragem dos sinais com resposta finita ao impulso (*Finite Impulse Response - FIR*). Dada a natureza deste tipo de algoritmo, que inclui um elevado número de operações aritméticas de soma e multiplicação, a sua implementação geralmente apresenta um elevado consumo de potência. Este fato se deve principalmente aos circuitos multiplicadores, que são responsáveis por uma parte significativa do consumo de potência global dos equipamentos da área de DSP. Desta forma, torna-se importante a aplicação de técnicas de baixa potência em circuitos da área de DSP. Estes métodos devem incluir a redução da atividade de chaveamento dos operadores aritméticos.

Na área de DSP, uma outra classe de algoritmos que utiliza uma grande quantidade de operadores aritméticos inclui o processamento de sinais a partir da transformada discreta de Fourier (*Discrete Fourier Transform - DFT*). Esta classe de algoritmos é largamente utilizada em aplicações que envolvem, por exemplo, processamento de imagens, comunicações, instrumentação, engenharia biomédica, etc.

Nestas aplicações, os sinais são processados a partir de algoritmos de transformada rápida de Fourier (*Fast Fourier Transform - FFT*), que processam eficientemente a Transformada Discreta de Fourier [BAA 2000]. Estes algoritmos são utilizados para reduzir o grande número de operações aritméticas envolvido.

Desta forma, a aplicação de técnicas de redução de potência a partir da redução da atividade de chaveamento neste tipo de algoritmo, torna-se importante dada a sua complexidade. Outro aspecto a ser destacado é o fato de que muitas das técnicas de redução de potência aplicadas ao algoritmo de filtros FIR podem ser utilizadas no algoritmo de FFT, tendo-se um amplo espectro de áreas de aplicação para os diversos circuitos implementados neste trabalho.

1.2 Objetivos

O objetivo deste trabalho de pesquisa é a aplicação de diferentes técnicas de redução de potência em circuitos voltados à área de Processamento Digital de Sinais. O estudo busca explorar a viabilidade de diferentes alternativas de arquiteturas específicas para algoritmos da área DSP, com aplicação de técnicas de redução de potência. Em particular, uma classe específica de algoritmos de filtros FIR e FFT são investigados.

O principal objetivo é reduzir o consumo de potência dos circuitos a partir da aplicação de técnicas de transformação voltadas à redução da atividade de chaveamento. Neste contexto, investiga-se a utilização de técnicas de codificação de dados em operadores aritméticos que são utilizados como módulos básicos nos circuitos de filtros FIR e FFT. Um dos principais enfoques é a exploração da correlação dos dados para a redução da atividade de chaveamento dos circuitos.

De fato, visto que a atividade de chaveamento é dependente da correlação entre dados sucessivos de entrada, o aumento da correlação resulta em geral na redução do consumo de potência dos circuitos. Neste trabalho, este aspecto é abordado a partir da apresentação de operadores aritméticos que utilizam o processo de codificação dos sinais como uma forma de reduzir as suas atividades de chaveamento. O objetivo desta tarefa não consiste somente da redução da atividade de chaveamento nos barramentos de dados, mas também a minimização da complexidade da lógica combinacional dos módulos. Investiga-se a utilização dos módulos aritméticos operando em um código diferente do Binário [COS 2001, COS 2001a], em arquiteturas específicas de filtros FIR e FFT.

Em geral existe menos correlação de sinal nos barramentos de dados comparado ao barramento de endereços, devido à propriedade da localidade de referência. No entanto, a metodologia proposta neste trabalho de doutorado visa mostrar que os aspectos da correlação em muitos casos pode ser significativo para a redução da capacitância chaveada também nos barramentos de dados.

Em algoritmos de filtros FIR e FFT é comum a utilização de circuitos multiplicadores que consideram operações com sinal. Neste contexto, o algoritmo de multiplicação Booth [BOO 51] vem sendo utilizado como forma de reduzir o consumo de potência destes operadores. Este algoritmo permite operações de multiplicação de operandos em complemento de 2 e apresenta como principal característica a redução do número de linhas de produtos parciais como forma de reduzir a atividade de transição ao longo do circuito, mantendo a regularidade do *array*.

No nosso trabalho, têm-se os mesmos objetivos do algoritmo Booth, ou seja, apresentar uma estrutura de multiplicador que permita a redução dos termos dos produtos parciais, enquanto mantém a regularidade de um multiplicador *array* convencional [HWA 79]. O objetivo é mostrar que a estrutura proposta pode ser mais eficiente do que o algoritmo Booth operando em complemento de 2 na representação Binária, ou mesmo em uma outra representação de código diferente do Binário.

Além da utilização dos módulos aritméticos de baixa potência nas arquiteturas específicas de algoritmos de filtros FIR e FFT, um outro foco do trabalho é a abordagem da exploração arquitetural dos circuitos que apontam para o aumento do desempenho e redução de *glitching* com a utilização de técnicas como *retiming e pipelining*.

Devido à natureza dos algoritmos investigados, que incluem operações de convolução com multiplicação de dados por coeficientes apropriados, também são investigadas técnicas que exploram a manipulação dos coeficientes, considerando aspectos de correlação, tais como a melhor forma de ordenação e particionamento.

1.3 Metodologia

A utilização de diferentes ferramentas dedicadas ao projeto é fundamental para a síntese e análise dos circuitos implementados, que envolve a transformação de descrições do nível de transferência de registradores (*Register Transfer Level- RTL*) dos circuitos para o nível estrutural e validação das técnicas de baixa potência.

Neste trabalho é usado um conjunto de ferramentas que se encontram disponíveis à comunidade científica. Entre estas, destacam-se as seguintes aplicações:

- Estimador de potência no nível lógico (*power estimate tool*) [TSU 95]. Esta ferramenta foi incorporada no ambiente de projeto de circuitos digitais SIS desenvolvido pela Universidade de Berkeley na Califórnia [SEN 92], fazendo parte integrante da distribuição oficial deste pacote largamente usado em Universidades e mesmo empresas em todo o mundo;

- Ferramentas para baixa potência. Estas ferramentas foram também desenvolvidas no ambiente SIS, estando disponíveis como opção para este sistema e envolvem:

  - Decomposição de máquinas de estado para baixo consumo [MON 98, MON 2000];
  - Precomputação [ALI 94, DEV 95, MON 96, HAS 98, MON 99];
  - *Retiming* de circuitos seqüenciais [MON 93, LAL 95, CAB 2000].

Os circuitos implementados neste trabalho estão descritos no formato *Berkeley Logic Interchange Format* (*BLIF*) utilizado pelo ambiente SIS. Este formato descreve uma hierarquia do circuito no nível lógico na forma textual. Um aspecto a ser destacado neste formato é a sua portabilidade, dada a difusão de uso do mesmo nos meios acadêmico e industrial.

No ambiente SIS, a dissipação de potência é calculada através da ferramenta *power estimate*, após a criação de uma rede simbólica do circuito [MON 96]. Nesta ferramenta, calcula-se o número esperado de transições de cada nó por ciclo de

relógio, baseado na probabilidade das entradas primárias. A estimativa de potência é efetuada de acordo com o modelo de atraso selecionado, que pode ser valores zero, unitário ou genérico (valor de atraso de cada porta lógica contido na biblioteca de portas lógicas).

Outro aspecto a ser destacado na estimativa de potência a partir da ferramenta *power estimate* é a possibilidade de utilização de diferentes tipos de sinais que podem ser aplicados às entradas dos circuitos. Estes sinais podem ser gerados externamente e fornecidos à ferramenta em um arquivo de entrada. Outra forma é a utilização de valores lógicos gerados aleatoriamente no próprio ambiente SIS, e introduzidos às entradas dos circuitos. Em ambos os casos, a quantidade de vetores de dados pode ser definida pelo usuário.

Para a verificação do consumo de potência de cada elemento do circuito, assim como o consumo associado à atividade de transição nos barramentos de dados, foi desenvolvido um editor com interface gráfica [POR 2000] para o ambiente SIS. O principal objetivo deste editor é a construção de um ambiente gráfico para manuseio de sistemas lógicos, incluindo a edição e a análise lógica instantânea e temporal do circuito. Esta ferramenta mantém a compatibilidade com a ferramenta de síntese lógica SIS, sendo portanto possível a troca de informações entre os ambientes e a utilização dos seus recursos como a ferramenta *power estimate* incorporada a este ambiente.

Embora a ferramenta *power estimate* do ambiente SIS possa calcular de forma eficiente o consumo de potência dos circuitos para modelos de atraso zero, unitário e genérico, o algoritmo envolvido no processo de cálculo de potência com simulação simbólica envolve uma grande carga computacional, como será mostrado posteriormente. Desta forma, a estimativa de potência utilizando esta ferramenta se torna inviável para circuitos de maior complexidade utilizados neste trabalho, tais como multiplicadores, filtros FIR e FFT, que utilizam palavras de dados de 16 bits.

Desta forma, está sendo utilizada a ferramenta de simulação no nível de transistores (*Switch Level Simulator - SLS*) do sistema OCEAN [GEN 89] com o objetivo de fazer a estimativa de potência nos circuitos implementados neste trabalho. A ferramenta SLS é um simulador no nível de chaves que pode ser utilizado para simular os comportamentos lógico e temporal de circuitos que podem ser descritos no nível de transistores tipo MOS, no nível de portas lógicas e no nível funcional.

Uma das principais características exploradas desta ferramenta neste trabalho é a sua capacidade de simulação de comportamento de circuitos tipo MOS com mais de cem mil transistores, sendo desta forma viável de ser utilizada para a estimativa da potência média consumida pelos circuitos implementados neste trabalho.

Deve-se destacar que além das ferramentas de análise de potência descritas anteriormente, outras ferramentas, tais como Synopsys [SOL02] e HSPICE [MET96] poderiam ser utilizadas neste trabalho. Entretanto, devido à simplicidade e portabilidade das ferramentas descritas, optou-se pelas suas utilizações.

1.4 Contribuições do Trabalho

A partir do estudo realizado para exploração das diferentes alternativas arquiteturais na classe de algoritmos de filtros FIR e FFT, com aplicação de técnicas de baixa potência, apresentam-se as seguintes contribuições:

- Projeto de diferentes arquiteturas específicas para os algoritmos de filtro FIR e FFT. A implementação de diferentes arquiteturas de circuitos incluindo a sua exploração arquitetural, considerando aspectos de área, atraso e consumo de potência servem como parâmetro para a escolha da melhor alternativa arquitetural destes circuitos para determinadas aplicações;

- Projeto de operadores aritméticos de baixa potência operando em um código diferente do Binário. A implementação destes operadores mostra que é possível a implementação de operadores aritméticos operando diretamente em um código diferente do Binário com aspectos de baixa potência, considerando a correlação dos sinais e a carga capacitiva nos barramentos;

- Projeto de circuitos multiplicadores em complemento de 2. Na área de DSP, uma das principais operações envolve a multiplicação de operandos em complemento de 2. Neste trabalho, apresenta-se uma nova arquitetura para multiplicação de operandos em complemento de 2. Esta arquitetura mantém a mesma regularidade de um multiplicador *array* convencional com aspectos de melhor desempenho e redução de potência. Esta arquitetura pode operar em números representados em código Binário ou em um código diferente do Binário;

- Metodologia de aplicação dos operadores do item anterior em circuitos voltados à área de processamento digital de sinais. A utilização dos operadores aritméticos propostos nas diferentes arquiteturas de circuitos voltados à área de DSP, é de fundamental importância para a identificação do percentual de redução de potência que pode ser obtido a partir da utilização destes operadores.

- Manipulação de coeficientes. Dada a natureza dos algoritmos de filtro FIR e FFT que apresentam a multiplicação de dados por convenientes coeficientes, uma das técnicas utilizada para redução de potência nestes algoritmos consiste na manipulação dos coeficientes de multiplicação com os dados de entrada para redução da atividade de chaveamento nos operadores aritméticos. Neste trabalho, aplica-se a técnica de ordenação de coeficientes proposta em [MEH 95] nas arquiteturas de filtro FIR e FFT totalmente seqüenciais. Uma extensão desta técnica é proposta no nosso trabalho, visando além da melhor ordenação, o melhor particionamento dos coeficientes. Esta técnica é aplicada em arquiteturas de filtros FIR e FFT semi-paralelas.

1.5 Organização do Trabalho

A estrutura deste trabalho é dividida em sete capítulos. Os principais fatores que influenciam no consumo de potência em circuitos CMOS são apresentados no Capítulo 2. O Capítulo 3 aborda as técnicas de otimização de potência nos diferentes níveis de abstração. O Capítulo 4 aborda as técnicas de otimização no nível arquitetural, objetivo alvo deste trabalho, destacando as técnicas que visam a redução da atividade de chaveamento e as técnicas que utilizam transformações na estrutura computacional dos circuitos para aumento de desempenho e redução da tensão de alimentação para redução de potência. O Capítulo 5 apresenta as técnicas de

redução de potência propostas neste trabalho, apresentando operadores aritméticos que operam diretamente com diferentes entradas codificadas. Estes operadores aritméticos são utilizados na exploração arquitetural de circuitos para processamento digital de sinais. O Capítulo 6 apresenta uma nova arquiteturas de multiplicador *array* em complemento de 2. A arquitetura mantém a mesma regularidade de uma arquitetura de multiplicador *array* convencional. O Capítulo 7 apresenta o estudo de casos com exploração arquitetural e aplicação de técnicas de redução de potência, que inclui os resultados de utilização dos operadores aritméticos com entradas codificadas. Neste capítulo é mostrado o resultado de aplicações de técnicas de redução de potência com exploração arquitetural a partir das arquiteturas dos algoritmos de filtro FIR e transformada rápida de Fourier (FFT). As principais conclusões do trabalho, bem como propostas para futuros trabalhos são finalmente apresentadas no Capítulo 8.

2 Consumo de Potência em Circuitos CMOS

O consumo de potência é um dos principais parâmetros no projeto de circuitos digitais. Neste contexto, a potência de pico é utilizada como um dos pontos de análise da confiabilidade dos circuitos. Entretanto, o fator mais crítico é o consumo médio de potência dos circuitos que operam durante um determinado tempo [CHA 95]. Nesta parte do trabalho, investiga-se as principais fontes de consumo de potência em circuitos CMOS. A tecnologia CMOS é particularmente abordada neste trabalho, por ser utilizada na maior parte dos projetos de circuitos digitais.

Existem três fontes principais de consumo de potência em circuitos CMOS em operação normal [WES 94]: (1) consumo de potência estática, causado pelas correntes de fuga e correntes *subthreshold*, (2) consumo de potência de curto-circuito, que ocorre devido à corrente direta da fonte de alimentação para o terra durante o processo de chaveamento de uma porta lógica e (3) consumo de potência dinâmica, que é resultado da carga e descarga das capacitâncias dos circuitos. As parcelas de potências dinâmica e de curto-circuito são as maiores responsáveis pelo consumo de potência global de um circuito integrado. Entretanto, a potência dinâmica aparece com um maior valor percentual contribuindo com 85% da parcela de consumo total [STA 97, POP 2000], enquanto que a potência de curto-circuito contribui com apenas aproximadamente 10% do valor de consumo de potência total. As correntes de fuga podem contribuir para um consumo de potência de até 5% da potência total.

Neste capítulo serão mostrados os principais aspectos que contribuem para as parcelas de consumo de potência em circuitos CMOS. Será apresentado um novo modelo de estimativa da potência de curto-circuito, proposto no âmbito deste trabalho. Este modelo é comparado ao modelo proposto em [VEE 84]. Também será abordada a metodologia de estimativa de parâmetros de área, atraso e consumo de potência nos circuitos implementados neste trabalho.

2.1 Consumo de Potência Estática

Em circuitos CMOS o consumo de potência estática é devido às correntes de fuga dos transistores e junções *pn* e correntes *subthreshold*. Para a tecnologia CMOS, este consumo de potência é baixo e normalmente não é considerado nas técnicas de otimização.

2.1.1 Exemplos de Consumo de Potência Estática em Circuitos não CMOS

Na Figura 2.1, um transistor de passagem metal óxido semicondutor tipo *N* (*N-type Metal Oxide Semiconductor - NMOS*) é conectado à porta de um circuito inversor CMOS. Com a aplicação da tensão V_{dd} aos terminais de porta e dreno do transistor de passagem NMOS, o nó *B* assume o valor V_{dd}-V_{tn}. Este valor de entrada leva a saída do inversor para o nível lógico baixo. Entretanto, o transistor metal óxido semicondutor tipo *P* (*P-type Metal Oxide Semiconductor - PMOS*) estará em estado de condução fraca, pois V_{gs}-V_t é aproximadamente igual a zero. Desta forma, nesta situação há condução de corrente estática da fonte de alimentação para o terra,

conforme mostra a Figura 2.1.

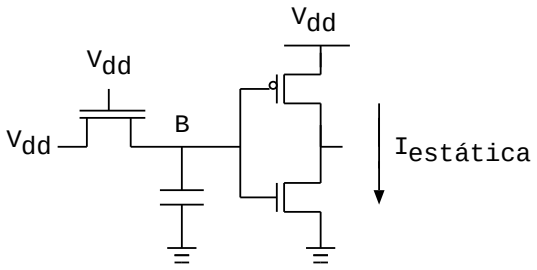

FIGURA 2.1 – Dissipação de potência estática em um inversor CMOS controlado por uma porta de passagem NMOS.

Embora para a tecnologia CMOS o consumo de potência estática seja reduzido e possa ser desprezado, em outras técnicas de circuitos tipo pseudo-NMOS (Figura 2.2) o consumo de potência estática é dominante, pois quando o sinal de entrada mantém o transistor NMOS conduzindo, uma corrente *DC* e a potência estática equivalente são consumidas, independente da frequência de chaveamento dos transistores [RAB 96].

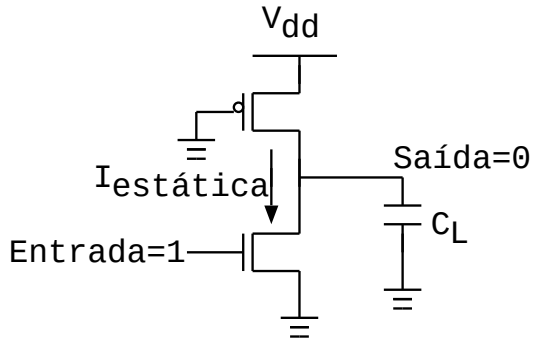

FIGURA 2.2 – Dissipação de potência estática em um inversor pseudo-NMOS.

Na técnica pseudo-NMOS o transistor tipo PMOS está sempre ativo. A situação do transistor NMOS depende da transição do sinal de entrada em sua porta. Se um nível lógico alto estiver presente, a saída da porta estará em nível lógico baixo e neste caso flui uma corrente estática direta da fonte de alimentação para o terra. O valor desta corrente depende da condutância *DC* série total dos transistores das redes PMOS e NMOS.

Deve ser observado que as figuras anteriores demonstram o consumo de potência estática para circuitos não CMOS. Nos circuitos CMOS, as situaçãoes mostradas não ocorrem e portanto, esta parcela de consumo de potência não está presente.

2.1.2 Potência Estática Devido às Correntes de Fuga

A componente de potência devido às correntes de fuga ocorre quando um transistor está no estado desligado e o outro transistor ativo carrega (*up/down*) o dreno em relação ao potencial de substrato [CHA 95]. Considerando um circuito inversor como exemplo com uma tensão de entrada alta, tem-se neste caso, o transistor NMOS conduzindo e o transistor PMOS no estado desligado, com a saída do circuito

em nível lógico baixo. Apesar do transistor PMOS apresentar-se no estado desligado nesta situação, sua tensão fonte-substrato será igual a -V_{dd}, pois a tensão de saída está em 0V enquanto que a tensão do substrato em relação ao transistor PMOS está em V_{dd} como pode ser observado na Figura 2.3.

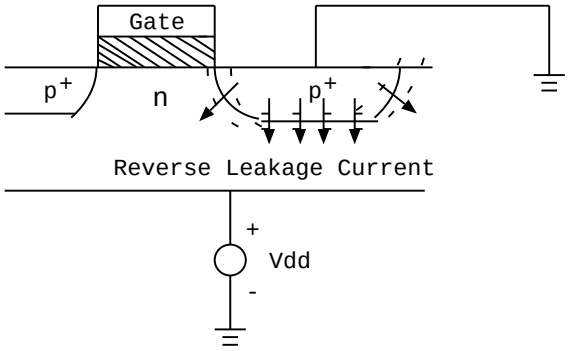

FIGURA 2.3 – Representação da corrente de fuga reversa.

De acordo com a Figura 2.3, a corrente de fuga é dada pela equação característica de um diodo de polarização reversa conforme é mostrado na Equação 2.1, onde I_S é a corrente de saturação reversa, V é a tensão do diodo e $V_T = KT/q$ é a tensão termal.

$$I_{leakage} = I_S(e^{\frac{V}{V_T}} - 1) \tag{2.1}$$

De acordo com [CHA 95], para um circuito integrado com um milhão de transistores, assumindo uma área de dreno média de $10\mu m^2$, a corrente de fuga total é da ordem de 25μA. Enquanto este valor de corrente representa uma pequena parcela do consumo total de potência na maior parte dos circuitos integrados, ela pode ser significativa para uma aplicação em que o circuito passe a maior parte do tempo em operações *standby*, pois esta componente de potência é sempre dissipada mesmo quando não há ocorrência de operações de chaveamento.

2.1.3 Potência Estática Devido à Corrente de Sublimiar

De acordo com [CHA 95], esta componente ocorre devido à difusão de portadores entre a fonte e o dreno quando a tensão porta-fonte (V_{gs}) excede o ponto de inversão fraca, mas ainda apresenta um valor menor do que a tensão de limiar (V_t). Neste regime, o transistor MOS comporta-se similarmente a um transistor bipolar e a corrente de sublimiar é exponencialmente dependente da tensão porta-fonte (V_{gs}) como mostra a Equação 2.2, onde K é um parâmetro dependente da tecnologia e n=1 + $\Omega t_{ox}/D$, onde t_{ox} é a espessura do óxido da porta, D é a largura da depleção do canal e Ω=$\epsilon_{si}/\epsilon_{ox}$.

$$I_{ds} = Ke^{(V_{gs}-V_t)/(nV_T)}(1 - e^{-\frac{V_{ds}}{V_T}}) \tag{2.2}$$

2.2 Consumo de Potência de Curto-Circuito

O consumo de potência de curto-circuito ocorre quando flui uma corrente diretamente da fonte de alimentação para o terra. Isto ocorre quando um circuito

CMOS estático é chaveado por um sinal de entrada com tempos de subida e descida não zero, tendo-se os transistores tipos PMOS e NMOS conduzindo simultaneamente por um curto intervalo de tempo.

A Figura 2.4 mostra o comportamento de um inversor, caracterizando a atividade de curto-circuito, assumindo que não há carga na saída. As formas de onda apresentadas para a tensão de entrada e corrente de curto-circuito foram obtidas através de simulação no *HSPICE* [MET 96], com um período do sinal de entrada de 10ns e uma rampa de 2ns [COS 99, COS 2000].

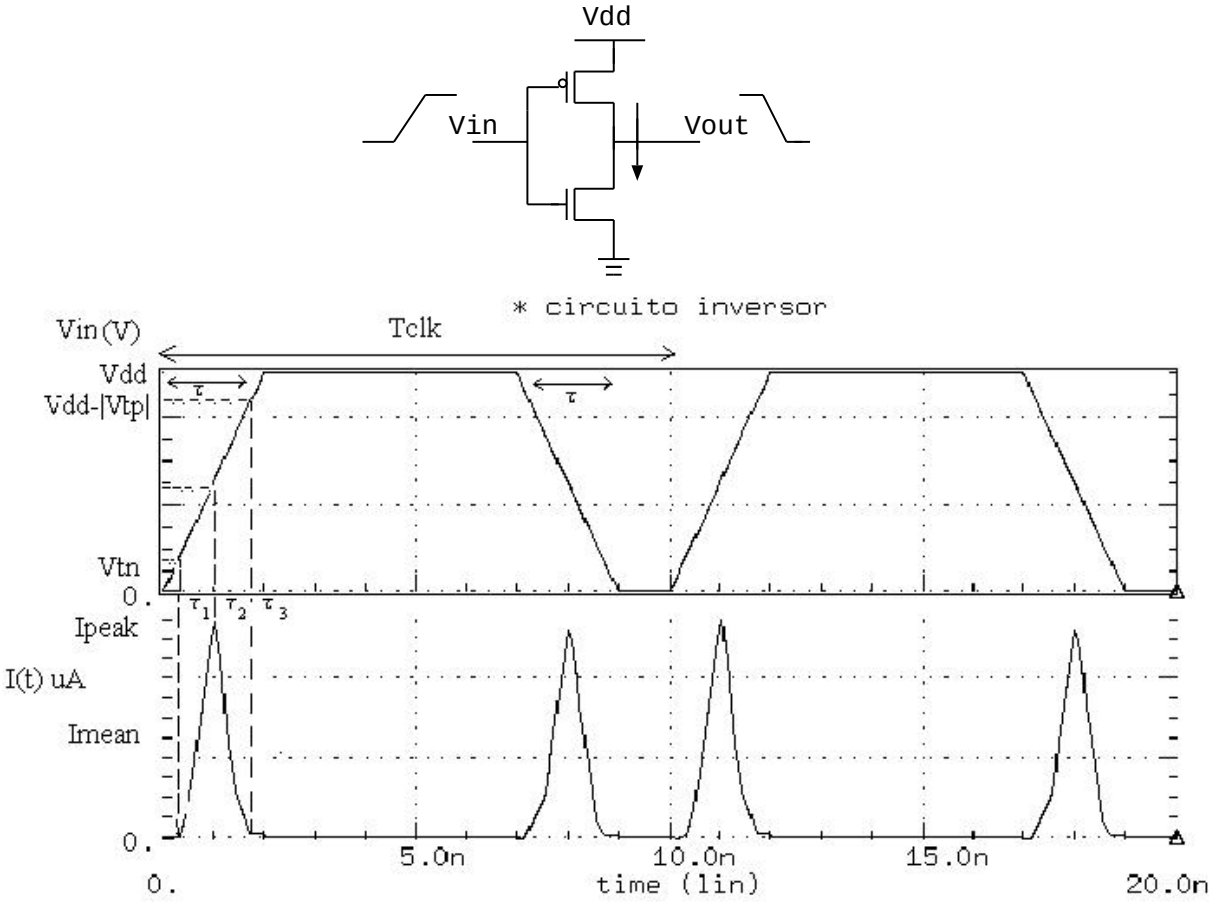

FIGURA 2.4 – Caracterização da corrente de curto-circuito em um inversor sem carga na saída.

No gráfico da Figura 2.4, observa-se que em uma transição de subida na entrada do inversor, o transistor NMOS começa a conduzir quando a tensão de entrada (V_{in}) é igual a tensão de *threshold* (V_{tn}) do NMOS. O transistor PMOS irá parar de conduzir quando V_{in} for igual a diferença da tensão de alimentação (V_{dd}) e a tensão de *threshold* do transistor PMOS (V_{tp}), ou seja, $V_{in} = V_{dd}$ - V_{tp}. Para as análises a seguir, assumimos V_{tn}=V_{tp}.

No intervalo de tempo de τ_1 até τ_2, o transistor NMOS estará na saturação visto que a tensão de saída será maior do que V_{in} - V_t. Nesta condição a corrente para o transistor NMOS é dada pela Equação 2.3 para uma condição $0 < I < I_{max}$ onde β é o parâmetro de transcondutância (expressa em A/V^2) efetiva do transistor NMOS.

$$I = \frac{\beta}{2}(V_{in} - V_t)^2 \tag{2.3}$$

A corrente média I_{mean} é utilizada para determinar a corrente de curto-circuito média que é drenada da fonte de alimentação. Uma aproximação para esta corrente média é dada por [VEE 84] e é representada na Equação 2.4.

$$I_{mean} = 2\frac{2}{T_{clk}}\int_{\tau_1}^{\tau_2} I(t)\,dt = \frac{4}{T_{clk}}\int_{\tau_1}^{\tau_2} \frac{\beta}{2}(V_{in}(t) - V_t)^2\,dt \tag{2.4}$$

Na Equação 2.4, o parâmetro V_{in} representa a tensão de entrada que é dada pela Equação 2.5 e τ_1 e τ_2 são os tempos de subida e descida respectivamente sendo representados nas Equações 2.6 e 2.7.

$$V_{in} = V_{dd}\frac{t}{\tau} \tag{2.5}$$

$$\tau_1 = V_t\frac{\tau}{V_{dd}} \tag{2.6}$$

$$\tau_2 = \frac{\tau}{2} \tag{2.7}$$

Inserindo os termos das Equações 2.5, 2.6 e 2.7 na Equação 2.4, a corrente média resultante é dada de acordo com a Equação 2.8, onde β é o parâmetro de transcondutância (expressa em A/V^2) efetiva do transistor NMOS, V_t é a tensão de *threshold* dos transistores e τ é a duração das rampas de subida e descida do sinal de entrada.

$$I_{mean} = \frac{\beta}{12V_{dd}}(V_{dd} - 2V_t)^3\frac{\tau}{T_{clk}} \tag{2.8}$$

A potência de curto-circuito é calculada pelo produto da tensão de alimentação pela corrente média e é dada de acordo com a Equação 2.9.

$$P_{sc} = V_{dd}I_{mean} = \frac{\beta}{12}(V_{dd} - 2V_t)^3\frac{\tau}{T_{clk}} \tag{2.9}$$

Existem diversos trabalhos para análise da potência de curto-circuito [VEM 94, HIR 96, HIR 98, ALV 98, TUR 98]. Parte destes trabalhos utiliza o modelo proposto por [VEE 84] dado na Equação 2.9 para a estimativa desta componente de potência.

No âmbito deste trabalho foi proposto um novo modelo para a estimativa da potência de curto-circuito [COS 99, COS 2000, COS 2000a]. Na abordagem proposta, observa-se que a Equação 2.9 fornece uma predição não muito precisa para P_{sc}, no limite de entradas rápidas, visto que P_{sc} depende de τ como mostrado na Figura 2.4. Na Figura 2.5 estão mostrados os dados de P_{sc} em valores normalizados, obtidos por simulação elétrica para um circuito inversor [COS 99]. Neste inversor tem-se $(W/L)_n$ =$(4/0,8)\mu$m e $(W/L)_p$ =$(9/0,8)\mu$m, para uma tecnologia CMOS 0,8μm da empresa *AMS* [AUS 98].

Observa-se pela Figura 2.5 que P_{sc} não se reduz a zero para uma entrada muito rápida (τ tendendo a zero). Na Figura 2.5, os valores de rampa de entrada são especificados pelo parâmetro de inclinação inverso T_{in} dado em ns/V. Deste modo, na escala logarítmica do gráfico da Figura 2.5 é possível observar que mesmo para uma entrada muito rápida T_{in}=1ps/V, a potência de curto-circuito não assume

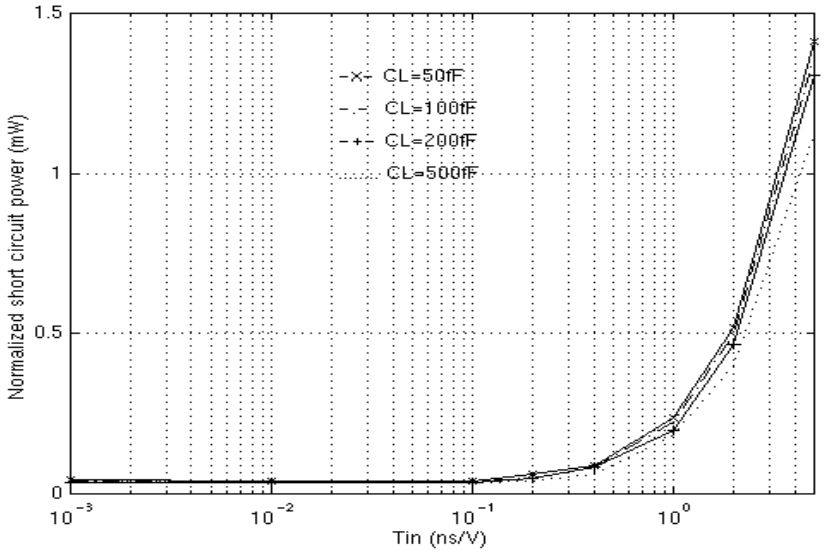

FIGURA 2.5 – Gráfico da potência de curto normalizada versus rampa de entrada logarítmica.

valor igual a zero. Isto ocorre porque a carga de inversão tem de ser drenada do canal dos transistores chaveando do estado *ON* para *OFF* na porta CMOS.

O modelo para potência de curto-circuito é representado pela Equação 2.10. As parcelas desta equação são representadas na Equação 2.11, que representa o modelo de potência de curto-circuito proposto em [COS 99]. Nesta equação, I_{off} é a corrente *subthreshold DC* e V_t é a tensão de *threshold*.

$$P_{scc} = P_{dc} + P_{sc} \tag{2.10}$$

$$P_{dc} = I_{off} V_{dd} \quad e \quad P_{sc} = P_o (V_{dd} - V_t)^{\alpha} \tag{2.11}$$

Em [COS 99] foi realizado um ajuste (*fitting*) da Equação 2.11, a partir de dados simulados para um circuito inversor, com V_{dd} variando de 1,5V a 5V, onde foram determinados valores de α = 2,2, 2,16 e 2,3 para transições de entrada (τ) iguais a 5ps, 50ps e 2ns respectivamente. Observa-se que V_t varia com V_{DS} devido ao efeito de redução da barreira de potencial induzida pela tensão de dreno (*Drain-Induced-Barrier-Lowering - DIBL*) que ocorre em transistores de canal curto usados em CMOS digital.

A redução linear de V_t foi modelada assumindo uma tensãão dreno-fonte média proporcional a V_{dd} e um parâmetro *ETA* (*SPICE DIBL*), de tal forma que a Equação 2.9 é reduzida a:

$$P_{sc} = P_o (V_{dd} - V_{to} + ETA \times V_{dd})^{\alpha} \tag{2.12}$$

Efeitos de canal curto resultam em valores de α menores do que 3 previsto por [VEE 84] na Equação 2.9. Em [COS 99], mostra-se que para o parâmetro *ETA* variando de 0 a 0,06 (para canais menores) o valor do expoente não varia significativamente, mostrando que o modelo da Equação 2.12 é mais preciso do que o proposto por [VEE 84].

2.3 Consumo de Potência Dinâmica

O consumo de potência dinâmica é resultado da carga e descarga das capacitâncias do circuito a partir da atividade de chaveamento. A componente de chaveamento dinâmica de dissipação de potência (P_{din}) em uma transição na saída de uma porta carregada por um capacitor C_L é dada de acordo com a Equação 2.13, onde A é a atividade do nó de saída, medida em eventos/segundo para uma carga/descarga completa. No caso de projetos síncronos, a atividade A não é simplesmente f (frequência), mas em geral uma probabilidade de atividade normalizada a (menor do que 1 para modelo de atraso zero) é computada como função da estatística de entrada e modelos lógicos, pois nem todos os nós mudam em um determinado ciclo de relógio.

$$P_{din} = \frac{1}{2} C_L A V_{dd}^2 = \frac{1}{2} a f C_L V_{dd}^2 \tag{2.13}$$

A componente de chaveamento de energia drenada da fonte de alimentação para uma transição $0 \rightarrow V_{dd}$ na saída de uma porta CMOS é dada por $C_L V_{dd}^2$, onde C_L é a capacitância de carga no nó de saída. A Figura 2.6 mostra um modelo de circuito destacando a componente de potência de chaveamento [CHA 95].

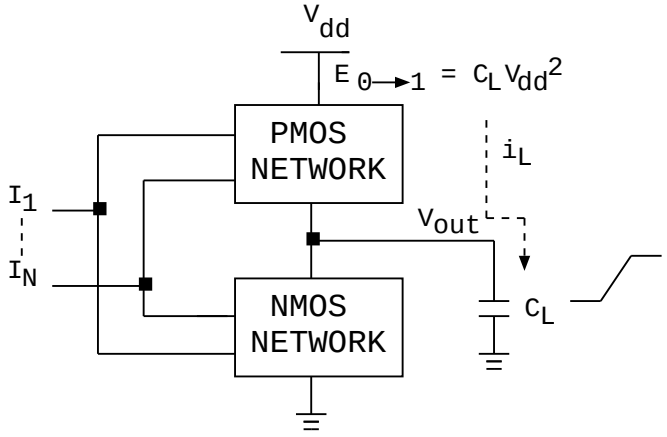

FIGURA 2.6 – Caracterização da potência dinâmica de uma porta CMOS.

Considerando-se uma transição $0 \rightarrow V_{dd}$ na saída do circuito, pode-se estabelecer a potência instantânea (ignorando a potência de curto-circuito) de acordo com a Equação 2.14, onde i_L é a corrente drenada da fonte de alimentação V_{dd} e que circula no capacitor C_L e é dada de acordo com a Equação 2.15.

$$P(t) = \frac{dE}{dt} = i_L V_{dd} \tag{2.14}$$

$$i_L = C_L \frac{dV_{out}}{dt} \tag{2.15}$$

A energia drenada da fonte de alimentação para uma transição $0 \rightarrow V_{dd}$ no nó de saída é dada de acordo com a Equação 2.16.

$$E_{0 \rightarrow 1} = \int_0^T P(t)\, dt = V_{dd} \int_0^{V_{dd}} C_L\, dV_{out} = C_L V_{dd}^2 \tag{2.16}$$

Para a transição na saída do circuito, a energia armazenada no capacitor de carga de saída, E_{cap} é dada pela Equação 2.17.

$$E_{cap} = \int_0^T P_{cap}(t)\,dt = \int_0^T V_{out} i_L(t)\,dt = \int_0^{V_{dd}} C_L V_{out}\,dV_{out} = \frac{1}{2} C_L V_{dd}^2 \qquad (2.17)$$

Conclui-se que metade da energia drenada da fonte de alimentação é armazenada no capacitor de carga, e metade da energia é dissipada na rede PMOS. Para uma transição $V_{dd} \rightarrow 0$ na saída, nenhuma carga é drenada da fonte de alimentação e a energia armazenada no capacitor ($1/2 C_L V_{dd}^2$) é dissipada na rede NMOS $pull - down$.

A Figura 2.7 mostra que a componente dinâmica normalizada é linearmente proporcional à capacitância de carga C_L e a extrapolação para C_L=0 é somente dependente dos fatores parasitas da porta, de acordo com a Equação 2.16 [COS 99]. Deste modo, a potência dinâmica (P_{din}) pode ser macromodelada, para um projeto baseado em células, de acordo com a Equação 2.18, onde C_L é a capacitância de carga extrínseca da porta e C_i é a carga intrínseca para uma célula lógica dada.

$$P_{din} = \frac{1}{2} a f V_{dd}^2 (C_i + C_L) \qquad (2.18)$$

Como pode ser visto na Figura 2.7, a rampa de entrada não afeta significativamente a componente de potência dinâmica, quando esta é controlada para ser rápida o suficiente em relação à saída. Para uma rampa de entrada infinita ($\tau \cong$0ns) até uma variação mais lenta na entrada τ=2ns (isto é, entrada T_{in}=0,4ns/V para V_{dd}=5V), a potência dinâmica tem como fator de controle fundamentalmente a carga/descarga do capacitor externo. Para rampas de entrada extremamente lentas (τ=25ns ou mais), observa-se a condição que tende ao regime de chaveamento adiabático [ATH 94, DEN 94, DIC 95]. Neste regime, pode-se ter uma componente dinâmica tão pequena quanto se queira, ao custo de circuitos extremamente lentos (como mostrado para uma rampa de entrada de 25ns na Figura 2.7). O regime adiabático não é considerado na modelagem da potência dinâmica, dado que neste regime, a principal componente de dissipação de potência é o curto-circuito.

2.3.1 Probabilidade de Chaveamento

Se em um circuito uma simples transição é realizada a cada ciclo de relógio na taxa f_{clk}, então a potência é dada por $\frac{1}{2} C_L {V_{dd}}^2 f_{clk}$. Entretanto, há casos em que a transição do sinal ocorre em diferentes taxas de frequência, tendo-se que considerar o valor do número de transições por ciclo de relógio ou o fator de atividade a de transição dos nós. Neste caso a potência média que corresponde ao número médio de transições de chaveamento em um período de tempo é dada por de acordo com a Equação 2.18, onde $a_{0\rightarrow1} = a_{1\rightarrow0} = a$. Na Seção 2.4 serão apresentadas técnicas de estimação do fator de atividade a.

Como os nós internos de um circuito podem também realizar transições, a atividade de transição tem de ser calculada para todos os nós do circuito e a potência total do circuito é dada de acordo com a Equação 2.19.

$$P_{total} = \sum_{i=1}^{n^o nos} \frac{1}{2} a_i C_i V_{dd}^2 f_{clk} \qquad (2.19)$$

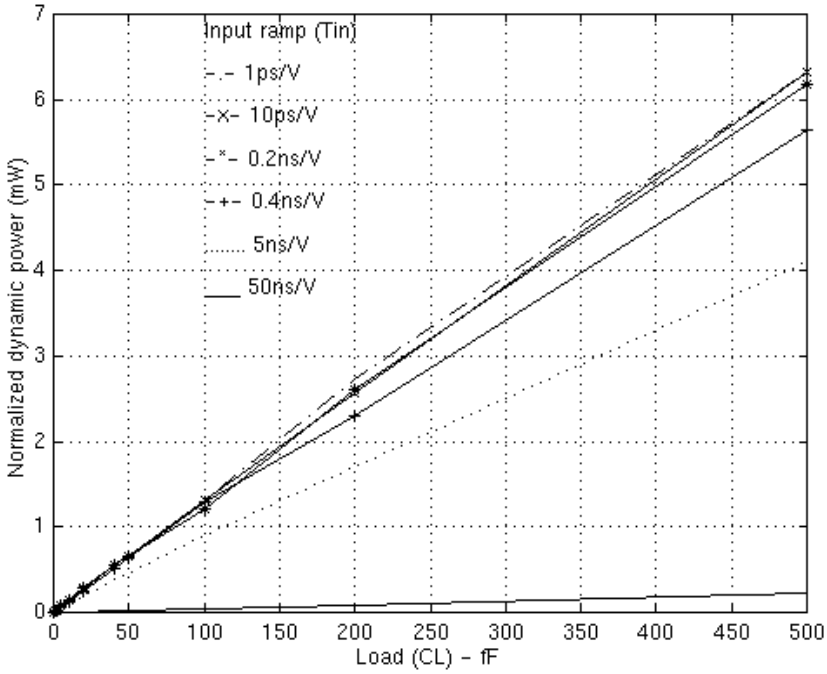

FIGURA 2.7 – Gráfico da variação das correntes dinâmicas com capacitâncias de carga e rampas.

2.3.2 *Glitching* em Circuitos CMOS Estáticos

Em circuitos CMOS o atraso de propagação entre portas lógicas ou blocos pode acarretar transições espúrias, ou seja, um nó pode exibir múltiplas transições em um ciclo de relógio antes de alcançar o nível lógico correto. Estas múltiplas transições são chamadas de *glitches* ou *hazards*.

A potência dissipada devido a estas múltiplas transições pode chegar a uma taxa de 15% a 20% da potência total em um circuito. Esta componente de potência é fortemente dependente da topologia do circuito e do padrão do vetor de entrada aplicado [FAV 95].

As principais contribuições de *glitches* para a dissipação de potência em um circuito digital são: geração na saída de uma porta e propagação através de uma porta (Figura 2.8) [FAV 95, RAB 96].

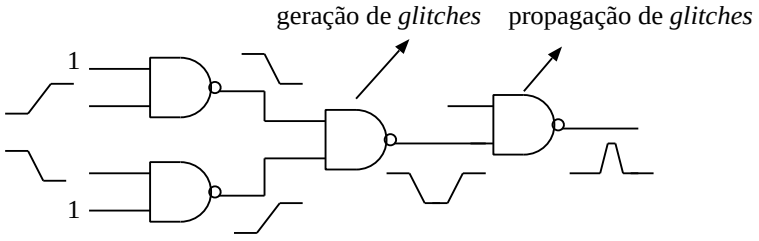

FIGURA 2.8 – Esquema de contribuição de *glitches* para dissipação de potência.

Observa-se que o instante de chaveamento do sinal de entrada de uma porta pode causar níveis lógicos falsos em sua saída, contribuindo para a geração de

glitches. A propagação destes sinais é uma função da profundidade lógica do circuito e, no pior caso, estas transições podem crescer de forma exponencial, contribuindo fortemente para o aumento do consumo de potência dos circuitos.

2.3.3 Componentes de Capacitâncias de Nós

As capacitâncias parasitas associadas aos circuitos são de fundamental importância na dissipação de potência e a sua formação está associada ao regime de operação de um transistor MOS. A Figura 2.9 mostra o modelo das capacitâncias parasitas de uma porta CMOS [CHA 95]. Deve-se observar que as mesmas capacitâncias mostradas na porta lógica composta pelos transistores $M1$ e $M2$ devem estar presentes na porta lógica composta pelos transistores $M3$ e $M4$. Entretanto, as capacitâncias destes transistores são omitidas na Figura 2.9 para simplificação do desenho.

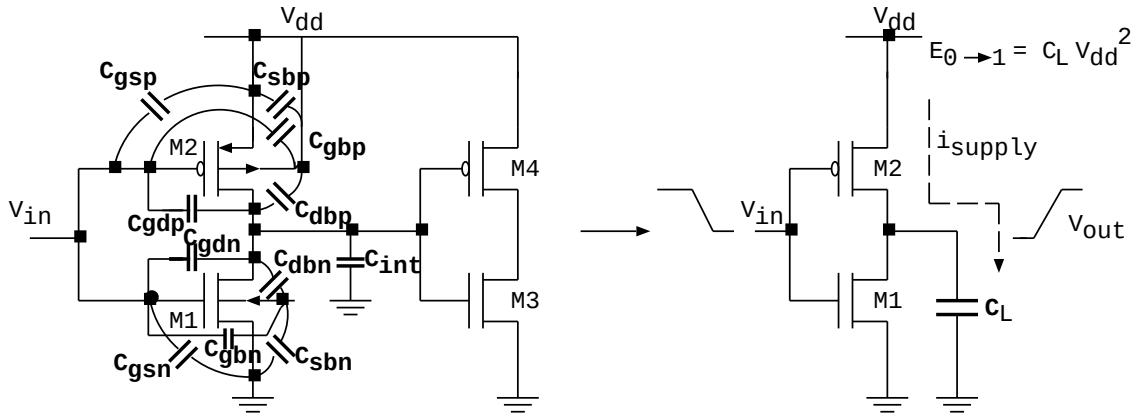

FIGURA 2.9 – Modelo das capacitâncias parasitas de uma porta CMOS.

A Figura 2.9 mostra que as diversas capacitâncias parasitas de um circuito podem ser representadas por uma única capacitância de carga (C_L), que é a soma de três componentes: capacitância de porta (C_g), capacitância de difusão (C_{diff}) (dividida em capacitância de sobreposição e capacitância de dreno e fonte) e capacitância de conexão (C_{int}). A seguir, apresenta-se cada uma destas componentes em maiores detalhes.

2.3.3.1 Capacitância de Porta

A capacitância de porta de um transistor MOS consiste de três componentes de capacitâncias: capacitância entre a porta e a fonte (C_{gs}), capacitância entre a porta e o dreno (C_{gd}) e capacitância entre a porta e o substrato (C_{gb}). O valor destas capacitâncias depende da região de operação do circuito, isto é, das polarizações V_G, V_D, V_S, V_B.

- Região de corte: nesta região não existe nenhum canal para passagem de cargas entre a fonte e o dreno e a capacitância total $(\epsilon_{sio2}/t_{ox})AREA$ aparece entre a porta e o substrato. A caracterização desta região ocorre quando $V_{gs}<V_t$,tendo-se $C_{gs}=C_{gd}=0$.

- Região linear (triodo): nesta região, supondo inversão fraca, resulta na formação de canal para passagem de cargas entre a fonte e o dreno. O efeito da camada de inversão é o de reduzir a valor baixo a componente C_{gb} ($C_{gb} \cong 0$). A simetria desta região indica que C_{gs}=C_{gd}= $((\epsilon_{sio2}/t_{ox})AREA)/2$. A caracterização desta região ocorre quando V_{gs}-V_t>V_{ds}.

- Região de saturação: nesta região, supondo inversão forte, tem-se que a camada de inversão se extingue na região de dreno, o canal é fortemente invertido, tendo-se uma estreita região de dreno. Nesta situação, C_{gd}<C_{gs}, enquanto que C_{gs} pode ser aproximado por $2/3(\epsilon_{sio2}/t_{ox})AREA$. A caracterização desta região ocorre quando V_{gs}-V_t<V_{ds}.

2.3.3.2 Capacitância de Sobreposição

No processo de fabricação dos transistores, ocorre uma extensão dos implantes de fonte e dreno além da borda do óxido de porta. Esta extensão abaixo do óxido é dada por uma quantidade x_d, como pode ser visto na Figura 2.10a [CHA 95]. Como resultado, o canal efetivo do transistor L_{eff} torna-se mais curto como pode ser visto na Figura 2.10b.

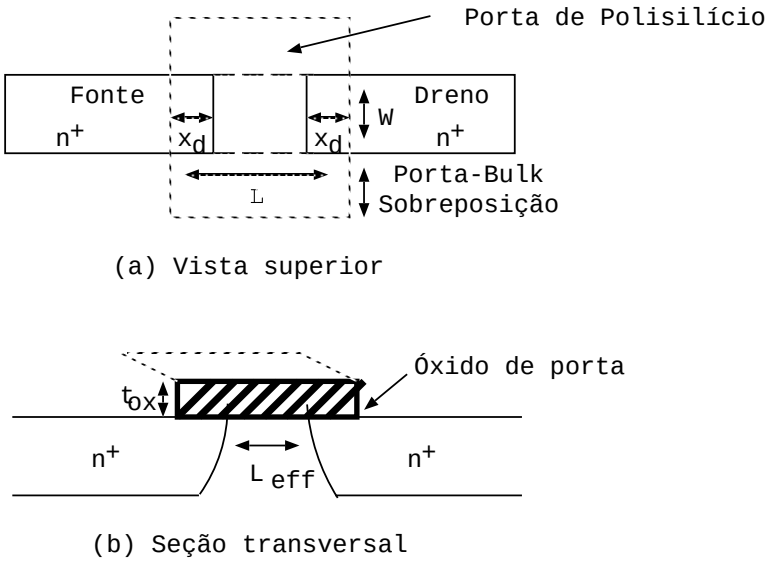

FIGURA 2.10 – Capacitância de sobreposição para um componente MOS.

A partir da característica de fabricação mostrada na Figura 2.10 para um transistor MOS, tem-se uma capacitância parasita entre porta e fonte (dreno), que é chamada capacitância de sobreposição. Esta capacitância é expressa na Equação 2.20, onde x_d é um parâmetro determinado pela tecnologia, e é tipicamente combinado com a capacitância de óxido, para produzir a capacitância de sobreposição por unidade de largura C_O.

$$C_{gs} = C_{gd} = C_{ox} x_d W = C_O W \tag{2.20}$$

2.3.3.3 Capacitância de Dreno e Fonte

A região de difusão (ou junção pn) apresenta uma capacitância relacionada ao substrato, cujo valor depende da tensão aplicada a esta região.

Esta capacitância é calculada a partir da área e perímetro dos transistores bem como de seus parââmetros de construção. No SPICE a capacitância de difusão de dreno ou fonte é calculada como mostra a Equação 2.21 [WES 94], onde C_J é a capacitância de junção por unidade de área, C_{JSW} é a capacitância de junção por unidade de perímetro, MJ é o coeficiente de nível da junção da parte inferior, $MJSW$ é o coeficiente de nível da junção da parte lateral (*side wall*), V_J é o potencial de junção, PB é a tensão de construção (0,4 - 0,8 volts) $AREA$ (AS ou AD) é a área da fonte ou dreno e $PERIMETRO$ (PS ou PD) é o perímetro da fonte ou dreno.

$$C_{drain} = [AREA * C_J(1 + \frac{V_J}{PB})^{-MJ}] + [PERIMETRO * C_{JSW}(1 + \frac{V_J}{PB})^{-MJSW}] \quad (2.21)$$

2.3.3.4 Capacitância de Conexão

A capacitância de conexão ou capacitância de roteamento entre camadas de polisilício e substrato pode ser aproximada usando um modelo de placas paralelas, conforme mostra a Equação 2.22, onde A é a área do capacitor de placas paralelas, t é a espessura do isolante, L é o comprimento da conexão, C' é a capacitância devido ao efeito de borda (*fringing*) e é expressa em pF por unidade de comprimento e ϵ_i é a permissividade do material isolante entre as placas.

$$C_{int} = \frac{\epsilon_i}{t}A + C'L \quad (2.22)$$

O modelo de placas paralelas ignora o efeito do campo elétrico que ocorre na borda do condutor, mas consiste em uma boa aproximação na avaliação do consumo de potência de circuitos CMOS.

As tabelas abaixo mostram valores característicos de parâmetros para as capacitâncias de um circuito inversor. A Tabela 2.1 mostra dados de transistores NMOS e PMOS para estimativa de valores típicos de capacitâncias para o inversor mostrado na Figura 2.9 na tecnologia AMS 0,8μm [AUS 98]. A Tabela 2.2 mostra os valores de capacitâncias dos parâmetros físicos para esta tecnologia. A Tabela 2.3 mostra os componentes de capacitância de carga para o inversor a partir da combinação dos parâmetros dos transistores com os parâmetros físicos.

TABELA 2.1 – Dados de transistores NMOS e PMOS para um inversor na tecnologia 0,8μm.

Transistores	W/L	AD	PD	AS	PS
NMOS	4/0,8	16P	9U	16P	12U
PMOS	9/0,8	36P	17U	36P	33U

TABELA 2.2 – Parâmetros físicos para a tecnologia 0,8μm.

Descrição	Símbolo	Valor
Capacitância de sobreposição *NMOS*	$CGDO(NMOS)$	0,34fF/μm
Capacitância de sobreposição *PMOS*	$CGDO(PMOS)$	0,34fF/μm
Capacitância de junção *bottom NMOS*	$CJ(NMOS)$	0,45fF/μm^2
Capacitância de junção *bottom PMOS*	$CJ(PMOS)$	0,6fF/μm^2
Capacitância de junção *side-wall NMOS*	$CJSW(NMOS)$	0,6fF/μm
Capacitância de junção *side-wall PMOS*	$CJSW(PMOS)$	0,54fF/μm
Capacitância de porta *NMOS*	$C_{ox}(NMOS)$	2,03fF/μm^2
Capacitância de porta *PMOS*	$C_{ox}(PMOS)$	2,03fF/μm^2

TABELA 2.3 – Componentes de capacitância de carga (C_L) para um circuito inversor.

Capacitor	Expressão	Valor(fF)
C_{gd1}	$CGDO\ W_n$	1,36
C_{gd2}	$CGDO\ W_p$	3,06
C_{db1}	Equação 2.21 (NMOS)	3,7
C_{db2}	Equação 2.21 (PMOS)	9,9
C_{g3}	$C_{ox} W_n L_n$	6,5
C_{g4}	$C_{ox} W_p L_p$	14,6

2.4 Metodologia de Análise de Circuitos Utilizados neste Trabalho

Para cada uma das fontes de consumo de potência de um circuito integrado, existem técnicas visando a sua estimativa de modo eficiente considerando os diversos fatores que definem os seus modelos. No presente trabalho, o enfoque é o consumo de potência dinâmica, pois este representa a maior parcela de consumo de potência de um circuito integrado.

A utilização de técnicas probabilísticas para estimativa de potência foram estabelecidas inicialmente para serem utilizadas em circuitos combinacionais [CIR 87, NAJ 90, NAJ 93], mas podem ser adaptadas para utilização em circuitos seqüenciais [MON 96a]. A precisão das técnicas probabilísticas está limitada pelos modelos de atraso e pela especificação dos sinais de entrada.

As técnicas probabilísticas exigem do usuário a especificação do comportamento típico das entradas dos circuitos e são baseadas nos seguintes critérios: (1) probabilidade de transição das entradas, (2) correlação temporal, (3) correlação espacial, (4) complexidade nas especificações das entradas, (5) potência individual consumida pelas portas, e (6) velocidade. A modelagem da estimativa de potência, considerando as características de complexidade nas especificações das entradas (associada à (1) probabilidade de transições), além dos aspectos de correlação (2) temporal e (3) espacial, podem tornar a implementação dos métodos de estimativa de potência uma tarefa de grande carga computacional. Desta forma, algumas técnicas ignoram estas características, tornando os resultados de estimativa de

potência menos precisos.

2.4.1 Ambiente de Análise de Potência

O procedimento completo de um projeto que envolve a análise de potência de um circuito, consiste em inicialmente estabelecer um nível de descrição da arquitetura do circuito a ser analisada. Esta descrição fornece à ferramenta de análise de potência uma lista de todos os diferentes módulos que têm de ser analisados bem como suas complexidades. Neste trabalho, a descrição dos circuitos, bem como a avaliação de potência em alguns circuitos, são realizadas no ambiente SIS [SEN 92]. A descrição dos circuitos é realizada no formato BLIF, cujo objetivo é descrever uma hierarquia do circuito no nível lógico na forma textual. Este formato é mapeado para uma biblioteca que contém a funcionalidade das portas lógicas, além das suas informações de capacitâncias e atrasos característicos. A estimativa de potência é realizada pela ferramenta *power estimate* [TSU 95]. Os dados de atividade dos sinais de entrada são fornecidos a esta ferramenta acoplada ao ambiente SIS, onde são associados os dados de atividades à uma biblioteca de valores de capacitâncias.

O consumo de potência dos circuitos é obtido na ferramenta *power estimate* do ambiente SIS, de acordo com a Equação 2.18. Os valores de potência encontrados levam em consideração valores de tensão e frequência de 5V e 20MHz respectivamente. Além disto, desde que não sejam fornecidos os valores de atividade, a ferramenta assume uma distribuição uniforme das entradas com probabilidade de chaveamento igual a 0,5 para todos os nós dos circuitos.

2.4.2 Análise de Potência Considerando *Glitches*

Algumas técnicas de estimativa de potência no nível lógico consideram o efeito de *glitches* nos circuitos a partir de simulação probabilística [TSU 93, DIN 98]. Alguns destes métodos consideram o efeito de correlação temporal entre as entradas do circuito, mostrando uma maior precisão nos resultados. Entretanto, este resultado é obtido ao custo de uma maior carga computacional.

O método de estimativa de potência utilizada no ambiente SIS é baseado em uma técnica de simulação simbólica que foi inicialmente apresentada em [GHO 92]. No ambiente SIS, é utilizado um modelo de atraso variável para a lógica combinacional no método de simulação simbólica, que computa as condições *booleanas* que causam *glitches* no circuito. Para cada porta lógica no circuito, a simulação simbólica produz um conjunto de funções *booleanas* que representam as condições para chaveamento em diferentes pontos no tempo. Desta forma, tendo-se a taxa de chaveamento nas entradas do circuito, pode-se calcular a probabilidade de chaveamento de cada porta lógica no circuito em algum instante de tempo particular. Os valores destas probabilidades são somados em todas as portas do circuito de tal forma a se obter a atividade de chaveamento em todos os pontos que correspondem a um ciclo de relógio.

No processo de utilização da técnica simbólica, é construída uma rede simbólica a partir da simulação do circuito lógico original. Uma rede simbólica representa um circuito lógico que tem as condições *booleanas* para todos os valores que cada porta na rede original pode assumir em diferentes instantes de tempo, dado um determinado par de vetores de entrada.

Se é utilizado um modelo de atraso zero, cada porta no circuito pode somente

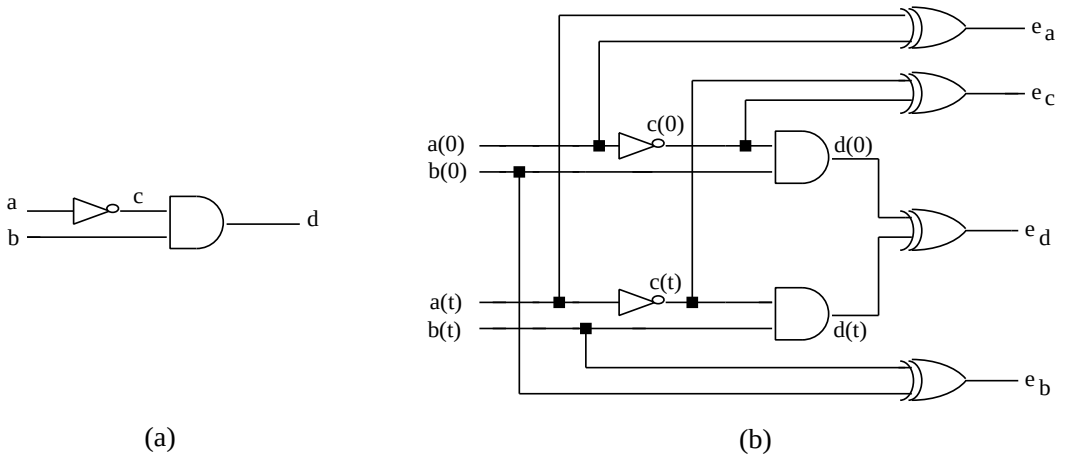

FIGURA 2.11 – Exemplo de aplicação da técnica simbólica para um modelo de atraso zero.

assumir dois valores diferentes correspondendo a cada vetor de entrada. Para este caso, a rede simbólica corresponde a duas cópias do circuito original, sendo uma cópia utilizada para o primeiro vetor de entradas e a segunda cópia utilizada com o segundo vetor de entradas. A Figura 2.11(a) mostra um exemplo de um circuito para simulação simbólica e a Figura 2.11(b) mostra a rede simbólica correspondente para um modelo de atraso zero [MON 96].

Como pode ser observado na Figura 2.11(b), além das duas cópias do circuito original da Figura 2.11(a) são utilizadas portas EXOR em cada par de nós. A saída de uma destas portas em nível lógico 1 indica que para este par de vetores de entrada, o correspondente nó no circuito original efetua uma transição. As entradas $a(0)$ e $b(0)$ correspondem ao primeiro vetor de entrada e $a(t)$ e $b(t)$ ao segundo. Se a saída e_c, por exemplo, assume valor lógico 1, então o sinal c no circuito original da Figura 2.11(a) fará uma transição para o par de vetores aplicado. O mesmo processo é repetido para as saídas e_a, e_b e e_d.

Para o caso de modelos de atrasos unitário e genérico, os nós de saída da porta de uma rede multinível podem apresentar múltiplas transições em resposta a uma seqüência de entrada de dois vetores. Em [MON 96] é apresentado o algoritmo de simulação simbólica e resultados de estimativa de potência considerando atrasos zero, unitário e genérico para diferentes circuitos.

Como pode ser observado, embora no ambiente SIS seja possível a estimativa de potência dos circuitos considerando diferentes modelos de atraso, de tal forma que os efeitos de *glitching* possam ser avaliados, a utilização de simulação simbólica em seu algoritmo de simulação requer cópia do circuito original para que as diferentes transições no circuito original possam ser avaliadas no tempo. Para circuitos maiores tipo multiplicadores e filtros digitais que são implementados neste trabalho, esta técnica requer uma grande carga em termos de memória, inviabilizando a sua utilização para a estimativa de potência.

2.4.3 Metodologia de Análises de Área, Atraso e Potência

Devido à grande carga computacional observada com a utilização da ferramenta *power estimate* para a estimativa de potência dos circuitos utilizados neste trabalho, utiliza-se então o ambiente SIS para efeitos de síntese e simulação dos

circuitos, além de cálculos de área e atraso. Entretanto, para a estimativa do consumo de potência dos circuitos, utiliza-se uma ferramenta no nível de transistores SLS [GEN 89]. Desta forma, os circuitos lógicos descritos em formato BLIF do ambiente SIS são convertidos para o formato da ferramenta SLS, conforme mostra o fluxo da Figura 2.12.

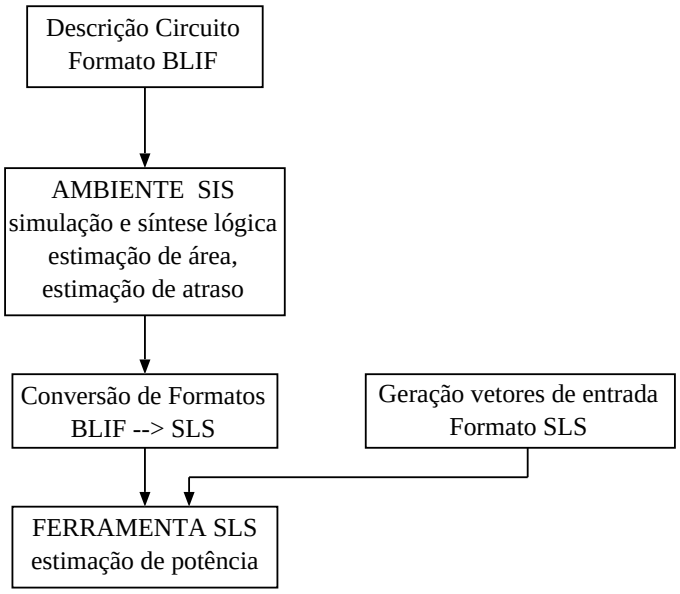

FIGURA 2.12 – Metodologia para estimativa de área, atraso e potência dos circuitos.

Como pode ser observado pela Figura 2.12, os valores de área e atraso dos circuitos são obtidos no ambiente SIS. Os valores de área são apresentados em termos do número de literais, onde 1 literal corresponde aproximadamente a 2 transistores. Os valores de atraso são obtidos a partir do modelo de atraso geral da biblioteca `mcnc`. Os valores de consumo de potência dos circuitos são obtidos a partir da ferramenta SLS após a conversão do circuito do formato BLIF para o formato no nível de transistores desta ferramenta. Os resultados de potência são obtidos após a aplicação de vetores de excitação às entradas primárias do circuito. A ferramenta SLS simula o comportamento do circuito ao longo do tempo. De acordo com os parâmetros do transistor e suas interconexões, o atraso é calculado para as diferentes transições no circuito. Neste cálculo, são utilizados (1) os tamanhos dos transistores, (2) as resistências de interconexão (R) e (3) valores de capacitância (C), para encontrar os tempos de subida e descida assim como valores estáveis dos sinais. Os tempos de subida e descida são computados pela interpretação de uma rede RC para cada nó do circuito. Desta forma, a dissipação média do circuito é calculada considerando o atraso genérico de cada porta lógica do circuito.

O conjunto de vetores de entrada para a ferramenta SLS é gerado a partir de um programa em linguagem de programação C que recebe como parâmetros de entrada o tipo de sinal com sua especificação (período de amostragem, defasagem, número de amostras), o tipo de codificação dos dados de entrada, e o comprimento da palavra de dados. Como saída deste programa, tem-se o sinal definido no formato da ferramenta SLS. Para esta ferramenta o bit com valor lógico 0 é representado como sinal baixo (*low -l*) e o bit com valor lógico 1 é representado como sinal alto (*high - h*).

O consumo de potência média dos circuitos é calculado a partir da transição dos sinais ao longo do circuito. Para o cálculo da potência é utilizada a expressão da energia $0{,}5CV_{delta}^2$ para cada oscilação da tensão V_{delta} que ocorre em um nó por um determinado intervalo de tempo. Para cada nó de entrada de uma porta lógica, a potência é calculada a partir do valor de energia dividido pelo comprimento do intervalo de tempo de oscilação da tensão. A potência média total é dada pela soma das parcelas individuais calculadas.

2.5 Resumo

Neste capítulo foram mostradas as principais fontes de consumo de potência em circuitos CMOS. A componente dinâmica corresponde a aproximadamente 85% do consumo da potência total de circuitos CMOS. Para esta componente de potência foram mostrados os principais fatores ligados a sua dissipação, tais como a probabilidade de atividade de chaveamento, a atividade de *glitching* e as componentes de capacitâncias de nós. As componentes de potência estática e de curto-circuito têm uma menor influência no consumo global de um circuito. Para a componente estática as correntes de fuga e correntes de *threshold* aparecem como principais responsáveis pela dissipação de potência. Para a componente de curto-circuito foi apresentado um estudo dos seus modelos de dissipação de potência, considerando o efeito da rampa do sinal de entrada. Foi apresentado um modelo de dissipação mais preciso do que o proposto em [VEE 84], tendo-se como principal observação que a potência de curto-circuito não reduz a zero mesmo para uma entrada muito rápida (rampa tendendo a zero). Para as diferentes fontes de consumo de potência em um circuito CMOS existem diferentes ferramentas voltadas às suas estimações de potência. No nosso trabalho, o enfoque é o estudo de potência dinâmica considerando aspectos de probabilidades de transição dos sinais. Desta forma, foi mostrada a metodologia de estimativa de potência para esta componente considerando as duas ferramentas de estimativa de potência utilizadas (SIS e SLS).

3 Técnicas de Otimização de Potência em Circuitos CMOS

Segundo [CHA 92a], o principal aspecto relacionado com a redução de potência é o estabelecimento da tensão da fonte em limites mínimos, sendo estes limites associados à tecnologia CMOS utilizada. Estes limites devem levar em consideração as mudanças nas condições de operação do circuito como a variação de temperatura ou ruído, além do aspecto do atraso, que tende a aumentar com a redução da tensão de alimentação.

Outro aspecto importante a considerar na redução do consumo de potência de circuitos CMOS é a atividade de chaveamento. Enquanto que a atividade de chaveamento é funcional, isto é, requerida para propagar e manipular informações, há uma quantidade substancial de atividades não úteis em circuitos digitais e que devem ser minimizadas em projetos de baixa potência. Um dos aspectos que contribui para estas atividades não úteis está relacionado com transições espúrias devido a atrasos de propagação de *glitches* (Seção 2.3.2). O outro aspecto está relacionado com as transições que ocorrem dentro de unidades que não são parte ativa em uma operação, ou cuja operação é redundante.

O projeto de circuitos integrados de baixa potência exige a combinação de técnicas nos níveis de tecnologia, circuito, lógico, arquitetura e algoritmos. Estas técnicas se baseiam principalmente em modelos de atraso simplificados e na modelagem de potência a partir do comportamento de sinais lógicos com probabilidades de transições. A precisão nos resultados destas técnicas está associada ao nível de abstração, conforme mostra o fluxo de projeto da Figura 3.1 [MEH 96a, LAN 98, POP 2000].

Como pode ser observado na Figura 3.1, no baixo nível de abstração (processo, circuitos, lógico), pode-se ter uma avaliação precisa do consumo de potência, pois se tem disponível detalhes de implementação dos circuitos (níveis de portas lógicas, transistores, *layout* dos elementos do circuito). Nos níveis mais altos de abstração (arquitetura, algoritmo e sistema), uma precisão relativa do consumo de potência é mais importante do que uma precisão absoluta, visto que nestes níveis de abstração o que se deseja realmente saber é se uma alternativa de projeto é melhor do que outra.

Os valores percentuais de redução de potência podem variar de acordo com o nível de abstração [POP 2000]. Como observado na Figura 3.1, no alto nível de abstração há maiores oportunidades de redução da dissipação de potência com valores de redução variando entre 2 e 20 vezes nos diferentes níveis de projeto. Isto ocorre devido ao maior grau de liberdade de projetos para baixa potência existente neste nível de abstração. Entretanto, nos níveis mais baixos de abstração, onde o grau de liberdade para projetos de baixa potência é mais reduzido, observam-se menores oportunidades de redução da dissipação de potência com valores entre 20 e 50%.

Neste capítulo serão abordados os principais aspectos de redução do consumo de potência nos diferentes níveis de abstração de projeto, com enfoque para os diversos graus de oportunidades para aplicação de técnicas de baixa potência.

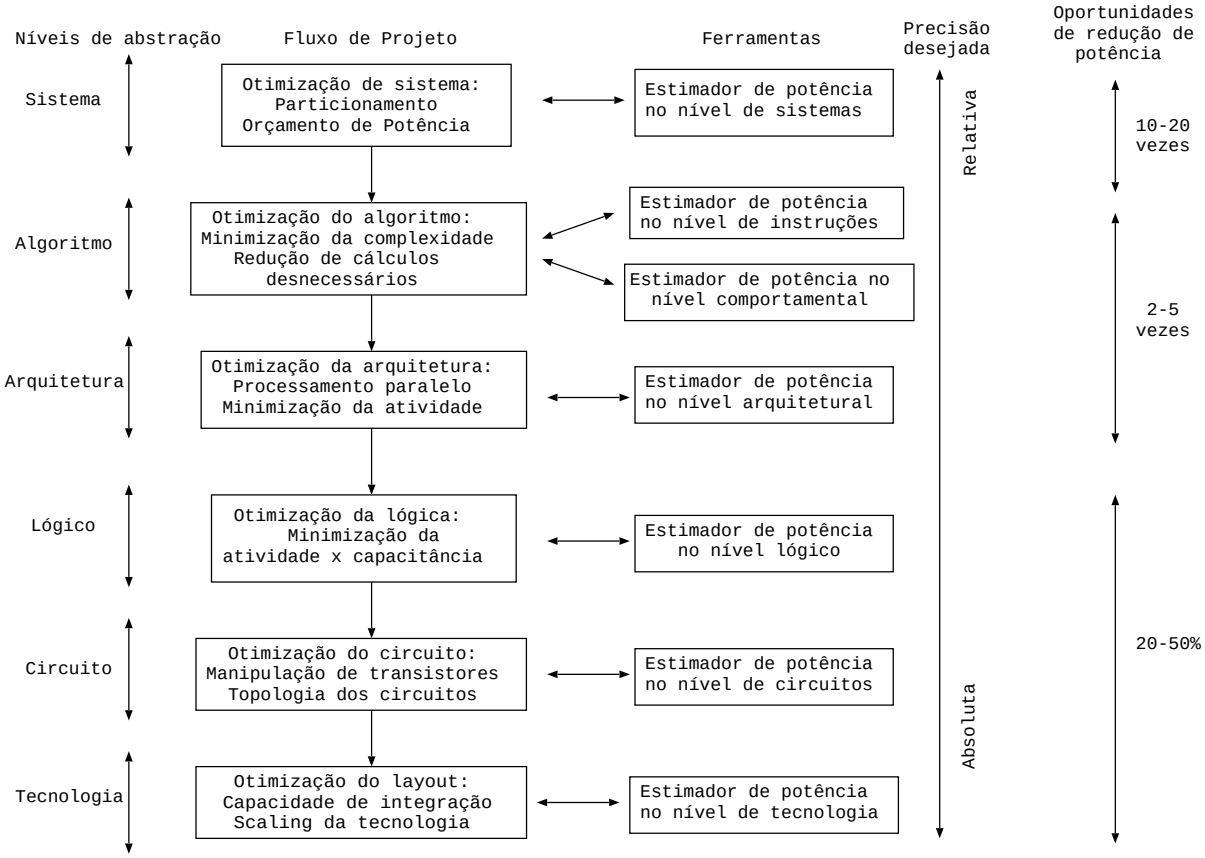

FIGURA 3.1 – Técnicas de otimização de potência para cada nível de abstração no fluxo de projeto.

3.1 Nível de Sistema

De acordo com [BEN 99], um sistema consiste de uma plataforma de *hardware* executando programas de *software* e é dividido em processadores, memórias e recursos de comunicação que constituem as fontes de dissipação de potência neste nível de abstração. As técnicas de otimização de potência neste nível são classificadas de acordo com a quantidade de flexibilidade para aplicação dos recursos de *hardware* disponível.

Os recursos de *hardware* estão relacionados com o tipo de processador utilizado para execução do processamento dos dados e estes podem ser divididos em microprocessadores de propósito geral, microcontroladores e processadores digitais de sinais e sistemas. Independente do tipo de processador utilizado, existe um grande grau de liberdade para projetos que incluem diferentes técnicas para redução do consumo de potência neste nível de abstração. Estas técnicas envolvem a otimização de memórias (utilização de memórias *cache*, por exemplo), particionamento de tarefas entre *hardware-software*, transformações no fluxo dados-controle, utilização de tensões de alimentação variáveis, minimização da potência de interfaces (barramentos) [BEN 99].

3.2 Nível de Algoritmo

Como pode ser observado na Figura 3.1, o fluxo de projeto que envolve o nível algorítmico é dividido em ferramentas no nível comportamental e no nível de instruções. De fato, a maioria dos estimadores de potência no nível comportamental assumem modelos arquiteturais correspondentes a implementações de *hardware* dedicado. Entretanto, também é possível realizar um dado comportamento em *software* em um processador de conjunto de instruções programável. Neste caso, um modelo de potência no nível de instruções se torna mais apropriado.

3.2.1 Nível Comportamental

Neste nível de abstração, muito pouco é conhecido sobre detalhes de implementação de um projeto. Neste caso, uma abordagem típica é a utilização de alguns estilos arquiteturais ou *templates* e produção de modelos de estimação de potência baseados nestes aspectos.

Os parâmetros de projeto desconhecidos e que têm de ser considerados incluem: configuração de memória, o número de acessos à memória, a arquitetura do barramento, o comprimento médio das interconexões, o número de operações no barramento, a complexidade do caminho de controle e a atividade das linhas de controle.

As técnicas de redução de potência neste nível de abstração estão principalmente relacionadas com o estudo de diferentes algoritmos para a realização de uma descrição de alto nível. Diretamente relacionados com este estudo estão os problemas de escalonamento e alocação de recursos.

3.2.1.1 Técnicas de Escalonamento

O objetivo dos algoritmos de escalonamento é associar a cada operação primitiva do grafo de fluxo de dados e controle (*Control-Data Flow Graph - CDFG*) um intervalo de tempo específico, de tal modo que a operação satisfaça a uma determinada restrição temporal. No caso de redução de potência, o escalonamento é realizado de forma a permitir que determinados módulos do circuito possam ser desabilitados em determinados passos do algoritmo [MON 96c]. Esta tarefa pode ser realizada a partir do bloqueio do sinal de relógio ou da própria redução da tensão de alimentação, desde que a conseqüente redução de desempenho não comprometa o fluxo de dados.

3.2.1.2 Alocação de Recursos

O objetivo da técnica de alocação de recursos é fazer corresponder a registradores e unidades funcionais, variáveis e operações do CDFG respectivamente, e especificar a interligação dos componentes em termos de barramentos e multiplexadores. O principal objetivo é minimizar a quantidade de módulos de *hardware* necessários e reduzir a quantidade de interconexão. Técnicas que consideram a dissipação de potência durante o processo de alocação também têm sido consideradas.

A técnica proposta em [RAG 94], direciona-se para a redução da atividade de chaveamento em registradores e unidades funcionais a partir de uma cuidadosa seleção de quais variáveis são apontadas para o mesmo registrador. A partir de simulação funcional, são obtidas estatísticas para cada variável, onde a partir destas estatísticas são obtidas as correlações entre as variáveis. O algoritmo seleciona

então para o mesmo registrador, as variáveis com alta correlação. Desta forma, o número de transições nos registradores e conseqüentemente nos módulos funcionais é minimizado.

Técnicas similares de redução da atividade de chaveamento considerando a alocação de recursos são descritas em [CHA 95] e [MUS 95]. Outras técnicas considerando a redução da atividade de chaveamento se direcionam somente para a alocação de registradores como em [RAG 95a]. Uma técnica para baixa potência que considera simultaneamente o escalonamento e a alocação de recursos é apresentada em [RAG 95].

3.2.2 Nível de Instruções

Uma outra abordagem para redução de potência no nível algorítmico, refere-se à otimização nas instruções de um código fonte. Neste nível, a potência pode ser estimada de acordo com o número de ciclos gastos por cada instrução e a sua corrente média. Esta estimativa de potência, torna-se importante em aspectos de projetos para baixa potência.

A otimização no nível de instruções está relacionada com o projeto de um algoritmo que seja bem mapeado para o *hardware* disponível e tenha eficiência nos aspectos de tempo e complexidade de armazenamento. Em termos de baixo consumo, as técnicas de otimização de potência neste nível estão relacionadas principalmente com a seleção de menor quantidade de instruções ou sequência de instruções.

No aspecto de seleção e ordenamento de instruções, o principal objetivo é a seleção de sequências de código que minimizem a potência ou energia através de diferentes instruções que possam ser compactadas em uma única instrução que opera em uma menor quantidade de ciclos. O ordenamento de instruções deve considerar o efeito do estado do circuito, ou a troca de operandos que resulte em menor atividade de chaveamento.

Em [TIW 96] é mostrado um exemplo com o agrupamento de duas instruções (*MUL*-multiplicação) e (*LAB*-transferência de dados de memória para registradores *A* e *B*) resultando em uma única instrução *MUL:LAB*. A instrução agrupada é aplicada ao programa de dois filtros (*IIR*4 e *LP-FIR*60), tendo-se reduções de energia de 26% e 47% respectivamente.

A troca de operandos é outra técnica no nível de instruções que pode resultar em ganhos significativos de consumo de potência. O objetivo desta técnica é a troca de operandos envolvidos em operações de unidades lógica e aritmética ULA ou operações com valores fracionários, de tal forma que minimize a atividade de chaveamento associada com as operações. Em [TIW 96], a técnica de troca de operandos foi utilizada em três exemplos de circuitos (*LP-FIR*60, *IIR*4, *FFT*2) utilizando as instruções do processador digital de sinais *TMS*320, com reduções de energia de 55,6%, 37,2% e 10,9% respectivamente em relação aos programas originais.

3.3 Nível Arquitetural

O presente trabalho de doutorado está voltado para este nível de abstração. Alguns pontos relativos às técnicas de otimização de potência utilizadas neste nível de abstração serão abordados nesta seção. Entretanto, o nível arquitetural será novamente abordado, em maiores detalhes, no próximo capítulo.

No nível arquitetural, uma das técnicas de baixa potência mais utilizadas é denominada técnica de bloqueio de sinal de relógio (*clock gating*). Esta técnica parte do princípio de que nem sempre todos os registradores presentes no circuito são atualizados a cada ciclo de relógio. Desta forma, torna-se necessário a identificação de quais partes do circuito podem ficar inoperantes por um determinado período de tempo. Se as condições que determinam a falta de ação dos registradores são observadas, pode-se obter a redução de potência através da inibição do disparo dos relógios nestes registradores de modo a desabilitá-los [MON 96a].

Outra classe de técnicas neste nível de abstração são baseadas na codificação de barramentos. De fato, as capacitâncias associadas aos barramentos costumam ser várias ordens de grandeza superiores às capacitâncias dos outros nós internos dos circuitos. Desta forma, em determinadas aplicações o consumo de potência nos barramentos pode ser muito significativo. Várias técnicas de redução da atividade de transição nos barramentos têm sido aplicadas. Estas técnicas serão abordadas no próximo capítulo.

3.4 Nível Lógico

As técnicas de otimização de potência no nível lógico têm tradicionalmente sido decompostas em técnicas independentes e dependentes da tecnologia. Em ambos os casos podem ser aplicadas técnicas para redução de área, atrasos, e dissipação de potência.

De acordo com [DEV 95], neste nível de abstração existem diversas técnicas de redução de potência, sendo a maioria delas direcionadas para a minimização da atividade de chaveamento dos circuitos combinacionais e seqüenciais. Algumas destas técnicas são aplicadas durante o processo de síntese lógica, como por exemplo, as técnicas que buscam o balanceamento de caminhos nos circuitos no sentido de reduzir as transições espúrias em circuitos lógicos combinacionais que representam, de acordo com [GHO 92], entre 10% e 40% da potência da atividade de chaveamento. O processo se encerra com o mapeamento tecnológico, onde se obtém uma descrição estrutural do circuito lógico, ou seja, uma rede (*netlist*) de dispositivos lógicos interligados.

No nível lógico, uma das técnicas de redução de potência mais difundidas atualmente é a técnica de pré-computação que consiste na introdução de portas de transmissão ou *latches* transparentes ao circuito para redução da sua atividade de chaveamento [DEV 95, MON 96a, MON 96b]. Neste caso, a idéia básica é a seleção e avaliação dos valores lógicos de saída do circuito em um ciclo de relógio anterior. No ciclo de relógio seguinte, estes valores são utilizados para desabilitação dos blocos não operantes no sentido de reduzir a atividade de chaveamento interna do circuito.

Uma outra técnica para redução da atividade de transição no nível lógico é a codificação de estados (*state assignment*) para baixa potência. Esta técnica utiliza a informação de probabilidades de transição de estado e da distância de *Hamming* para cálculo do número médio de chaveamentos dos bits de estado numa máquina de estados finita (*Finite State Machine - FSM*) [VEE 95]. A distância de *Hamming* é o número total de bits diferentes entre dois vetores binários de mesmo comprimento.

O posicionamento de registradores em pontos estratégicos do circuito tem sido utilizado como técnica para baixa potência conhecida por *retiming*. Esta técnica tem

por objetivo o reposicionamento de *flip-flops* em circuitos seqüenciais, mantendo-se seu comportamento funcional externo. Desta forma, tem-se a modificação da atividade de chaveamento em sinais internos de um circuito com impacto direto na dissipação de potência média. A utilização de *retiming* para minimizar a atividade de chaveamento é baseada na observação de que a saída de *flip-flops* em circuitos seqüenciais pode apresentar menos transições do que as suas entradas devido à redução da atividade de *glitching* [MON 96a].

3.5 Nível de Circuitos

A componente dominante do consumo de potência para um circuito CMOS é proporcional ao quadrado da fonte de alimentação. Desta forma, operações de circuitos em tensões mais baixas torna-se o ponto central para a minimização de energia consumida por operação [CHA 95]. Entretanto, a simples operação de reduzir a tensão de alimentação faz com que os elementos de circuitos individuais operem de uma forma mais lenta e isto tem de ser compensado por projetos de arquiteturas apropriados.

O produto potência-atraso pode ser interpretado como a quantidade de energia gasta em cada evento de chaveamento (ou transição). Se é levada em consideração somente a componente dinâmica da dissipação de potência, então a energia é dada de acordo com a Equação 3.1 [CHA 92a], onde $C_{effective}$ é a capacitância efetiva chaveada para realizar uma operação e é dada de acordo com a Equação 3.2, onde C_L é a capacitância de carga e a é o fator de atividade.

$$energia\_por\_transicao = \frac{P_{total}}{f_{clk}} = C_{effective} V_{dd}^2 \tag{3.1}$$

$$C_{effective} = aC_L \tag{3.2}$$

Como pode ser observado pela Equação 3.1, a energia por transição ou equivalentemente o produto potência-atraso é proporcional a V_{dd}^2 exibindo uma dependência quadrática. Entretanto, a simples solução de reduzir V_{dd} para projetos de baixa potência tem um custo. O custo a ser pago é a penalidade em velocidade para uma redução de V_{dd}, com o aumento do atraso [CHA 95], como mostra a Equação 3.3. Na Equação 3.3, μ e C_{ox} são parâmetros de tecnologia (mobilidade dos portadores e capacitância de óxido), W e L são a largura e o comprimento do transistor respectivamente e V_t é a tensão de limiar.

$$T_d = \frac{C_L V_{dd}}{I} = \frac{C_L V_{dd}}{\frac{\mu(C_{ox})}{2}\frac{W}{L}(V_{dd} - V_t)^2} \tag{3.3}$$

Algumas técnicas têm apontado para a redução da dissipação de potência a partir da manipulação de transistores. Uma destas técnicas é a redução da potência pelo reordenamento dos transistores e distribuição das entradas [CON 97]. O objetivo desta técnica é encontrar o melhor ordenamento de transistores conectados em cada porta e distribuição das entradas, no sentido de minimizar o atraso e/ou a dissipação de potência.

A redução de potência pela topologia dos circuitos tem também sido empregada como técnica de redução de potência neste nível de abstração. Nesta técnica, a

idéia central é a escolha de topologias de circuitos apropriadas para a implementação de várias funções lógicas e aritméticas. De acordo com [CHA 95], esta escolha passa por etapas de implementação como a utilização de lógica dinâmica ou estilos de implementação lógica com a utilização de transistores de passagem [YAN 90, TAK 97]. De acordo com [CHA 95], uma desvantagem apresentada em relação à utilização de lógica dinâmica se refere à necessidade de operação de pré-carga que pode resultar em um alto fator de atividade de transição, tendo-se que utilizar lógica adicional para compensação.

3.6 Nível de Tecnologia

A otimização de vários aspectos das tecnologias de fabricação CMOS pode resultar em reduções significativas de potência em circuitos integrados. Este aspecto está principalmente relacionado com a crescente capacidade de integração em um único *chip* (Capítulo 1). Uma maior integração permite a redução das capacitâncias parasitas (Seção 2.3.3), por efeito da redução do comprimento das ligações entre os componentes do circuito resultando em uma redução do consumo de potência. De fato, os avanços nas tecnologias de fabricação dos circuitos integrados vêm permitindo a redução no tamanho dos componentes, afetando parâmetros de desempenho dos circuitos como área, velocidade e consumo de potência. De acordo com [WES 94], se o fator de escala de uma tecnologia de fabricação é reduzido por um fator α=2 (0,5μm reduzido para 0,25μm por exemplo), então para uma redução da tensão de alimentação por um fator $1/\alpha$, a área é reduzida por um fator $1/\alpha^2$, o atraso é reduzido por um fator $1/\alpha$ e o consumo de energia é reduzido por um fator $1/\alpha^3$. Deve-se entretanto observar que com a redução do atraso, os circuitos podem operar em uma frequência mais elevada o que aumenta o consumo de potência.

Com o *scaling* da tecnologia, um dos principais fatores de redução de consumo de potência é a redução da tensão de alimentação a limites mínimos. Neste contexto, tem sido dada uma especial atenção para a produção de circuitos CMOS de ultra baixa potência, onde a tensão de alimentação é reduzida a dezenas de milivolts [BUR 91, BUR 94]. Entretanto, para a manutenção de bom desempenho do circuito nestes níveis de redução da tensão de alimentação, a tensão de *threshold* dos transistores também deve ser reduzida [KIM 2002]. Isto se deve ao fato de que para valores de tensão de alimentação próximos à tensão de *threshold*, o atraso tende a aumentar como mostra a Equação 3.3. A redução da tensão de *threshold* requer uma mudança no processo de fabricação e parâmetros como ruído e variação da temperatura devem ser considerados no ajuste da tensão de *threshold* para a manutenção do desempenho dos circuitos e redução dos seus consumos de potência.

3.7 Resumo

Este capítulo abordou as técnicas de otimização de potência utilizadas nos diferentes níveis de abstração. Como pôde ser observado, existem maiores oportunidades para aplicação de técnicas de redução de potência nos níveis mais altos de abstração (arquitetural, algorítmico, sistemas). Isto devido ao maior grau de liberdade para projetos de baixa potência. Nestes níveis de abstração, a liberdade do pro-

jetista está na exploração de arquiteturas para baixa potência ou mesmo na simples escolha do melhor algoritmo considerando as restrições de quantidade de unidades de execução e caminho crítico. Um melhor orçamento de potência para sistemas, considerando o grau de flexibilidade para aplicação dos recursos de *hardware* disponível, é outro enfoque presente no alto nível de abstração. Nos níveis mais baixos de abstração, observou-se a utilização de técnicas nos níveis lógico, transistores e tecnologia. No nível lógico foram abordadas técnicas voltadas à redução da profundidade lógica e redução na atividade de chaveamento de circuitos lógicos e combinacionais, tais como pré-computação, balanceamento de caminhos nos circuitos e *retiming*. As técnicas utilizadas em transistores envolvem desde a sua simples troca de posição em uma rede de circuito ou mesmo seu melhor dimensionamento. O *scaling* da tecnologia foi outro aspecto abordado no baixo nível de abstração, pois a otimização de vários aspectos das tecnologias de fabricação CMOS pode resultar reduções significativas de potência em circuitos integrados. No próximo capítulo serão aprofundadas as técnicas de otimização de potência no nível arquitetural.

4 Redução de Potência no Nível Arquitetural

Como abordado no Capítulo 3, existe intensa pesquisa em métodos de projeto para baixa potência, considerando os diferentes níveis de abstração da descrição do projeto. Pesquisas iniciais se direcionaram para os níveis de circuitos e portas lógicas. A tendência nos dias de hoje é a investigação de técnicas em níveis mais altos de abstração de projetos, pois tem sido reconhecido que o potencial para redução de potência é muito maior nestes níveis.

Este trabalho tem como foco técnicas de otimização de potência no alto nível de abstração, e em particular, no nível arquitetural e de transferência de registradores (RT).

No nível arquitetural, o circuito é descrito em termos de blocos funcionais, destacando os diferentes níveis de complexidade e a forma como são interligados, como mostra a Figura 4.1 [LAN 98].

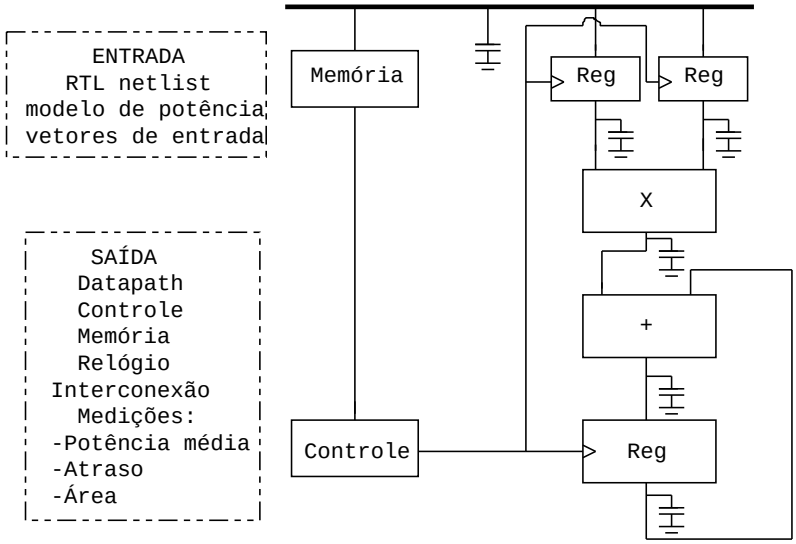

FIGURA 4.1 – Ilustração das primitivas do nível arquitetural.

Pode-se observar pela Figura 4.1 que no nível arquitetural existem dois principais componentes de circuito. O primeiro componente é o *datapath*, que é composto por módulos funcionais (somadores, multiplicadores, registradores, memória), que têm as funções de efetuar as operações e manter os resultados das operações. O segundo componente é o controlador, que gera a seqüência de sinais de controle para o *datapath*, indicando as operações que devem ser executadas por cada unidade funcional e a seqüência das mesmas.

No nível arquitetural, diversas técnicas podem ser aplicadas para a redução do consumo de potência. A principal atuação neste nível envolve a aplicação de técnicas que possam reduzir a capacitância efetiva chaveada, evitando-se perdas computacionais. As perdas computacionais representam o desperdício de potência relacionado às transições temporárias, (*glitches*, ou mesmo transições lógicas intermediárias ocorridas durante a operação aritmética/lógica), ou às transições lógicas em unidades operativas inativas nos ciclos de computação de outras unidades ativas em paralelo. Estas formas de perdas computacionais são provenientes de ações

desnecessárias no circuito que podem ser evitadas pela aplicação de técnicas específicas. As técnicas para evitar estas perdas no nível arquitetural incluem a preservação da correlação dos dados [MEH 96a]. Neste aspecto, deve-se considerar que a atividade de chaveamento é dependente da correlação entre dados sucessivos e o aumento da correlação pode resultar em uma maior redução de potência. Outro ponto importante neste nível de abstração é a aplicação de unidades específicas, pois estas consomem menos potência do que unidades de propósito geral, devido à estrutura mais simples e o controle reduzido para suporte de programação.

Nesta parte do trabalho serão abordadas estratégias que têm por objetivo a redução da atividade de chaveamento e técnicas de transformação, que devem explorar alternativas arquiteturais que atinjam níveis mínimos de consumo de potência.

4.1 Técnicas de Redução da Atividade de Chaveamento

A redução da atividade de chaveamento é um dos principais aspectos relacionados com a redução do consumo de potência em circuitos CMOS, visto que estes circuitos dissipam pouca energia se não estiverem chaveando [CHA 95, MUS 95, MEH 96a, STA 97]. A redução da atividade de chaveamento se baseia em técnicas que se estendem a aspectos de correlação, que podem existir entre valores de uma seqüência temporal de dados. Esses aspectos estão relacionados com o grau de compartilhamento entre os recursos de *hardware* utilizados, a escolha da representação dos dados (ou operandos), exploração de correlações de sinais e minimização de transições de *glitching*.

Em sinais completamente aleatórios a probabilidade de transição de qualquer bit do barramento é de 50%. No entanto, em processamento de sinais existe uma grande correlação entre valores consecutivos e esta correlação leva a que haja, em geral, apenas um número reduzido de bits diferentes entre valores consecutivos no barramento.

4.1.1 Compartilhamento de Recursos

O objetivo da abordagem desta técnica é mostrar como o grau de compartilhamento de recursos pode influenciar na atividade de chaveamento e conseqüentemente na dissipação de potência. Visto que em uma implementação de circuito existe uma escolha entre uma arquitetura totalmente paralela ou multiplexada no tempo, dependendo do *throughput* da aplicação, uma importante questão é saber qual arquitetura pode resultar em uma menor atividade de chaveamento.

Dependendo do tipo de aplicação, uma maior ou menor quantidade de *hardware* pode ser utilizada para realizar uma determinada tarefa. Neste aspecto, existe um relacionamento em que operações mais rápidas podem ser realizadas ao custo de uma maior quantidade de recursos. De outra forma, o *hardware* pode ser reutilizado tanto quanto possível comprometendo entretanto o desempenho do circuito.

A Figura 4.2 mostra um exemplo utilizando a técnica de compartilhamento de recursos a partir da implementação de dois circuitos contadores, cujas saídas são ligadas em barramentos compartilhados no tempo [CHA 95]. Na primeira implementação, utilizam-se barramentos paralelos e na segunda barramentos multiplexados no tempo.

Para a arquitetura de barramentos paralelos mostrada na Figura 4.2(a), tem-

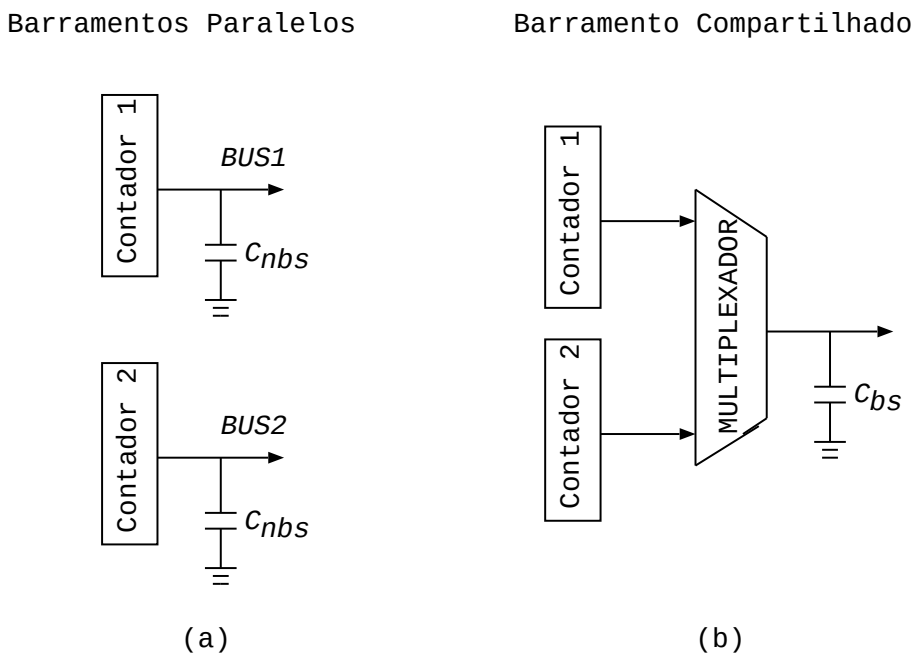

FIGURA 4.2 – Exemplo de estruturas de compartilhamento de barramentos.

se dois barramentos separados operando na mesma freqüência f. Considerando uma tensão fixa V, a potência consumida nesta arquitetura é dada pela Equação 4.1, onde C_{nbs} (*no-bus sharing*) representa a capacitância física por bit de *BUS1* e *BUS2* e a_i é a atividade de transição para o bit i.

$$P_{no\text{-}bus\ sharing} = \frac{1}{2}(\sum_i a_i)C_{nbs}V^2 f + \frac{1}{2}(\sum_i a_i)C_{nbs}V^2 f = (\sum_i a_i)C_{nbs}V^2 f \quad (4.1)$$

Deve-se observar que a atividade é dada em função da saída dos circuitos contadores, tendo-se neste caso: $\sum_i a_i = (1 + 1/2 + 1/4 + 1/8 + \cdots + 1/128)$ para contadores de 8 bits. Considerando que o bit menos significativo do contador chaveia a cada ciclo de relógio, tem-se que cada barramento terá na média um total de aproximadamente duas transições por ciclo de relógio.

Para a implementação de barramento compartilhado mostrada na Figura 4.2(b), tem-se um único barramento com uma capacitância C_{bs} (*bus sharing*), que é chaveada no dobro da freqüência f. Neste caso, o consumo de potência desta implementação é dado de acordo com a Equação 4.2.

$$P_{bus\text{-}sharing} = \frac{1}{2}(\sum_i a_i)C_{bs}V^2 2f = (\sum_i a_i)C_{bs}V^2 f \quad (4.2)$$

Na implementação de barramento compartilhado, a atividade em função da saída dos contadores é dada por: $a_i = 2(1 + 1/2 + 1/4 + 1/8 + \cdots + 1/128)$, tendo-se um número médio de aproximadamente 4 transições de barramento por ciclo de relógio.

Deve-se notar que a capacitância de barramento compartilhado (C_{bs}) será maior do que a capacitância de barramentos paralelos (C_{nbs}). Isto se deve a maior

quantidade de interconexões necessárias para a implementação de barramento compartilhado, tendo-se desta forma, um maior comprimento dos fios.

O exemplo da Figura 4.2 mostra que a implementação de barramentos paralelos apresenta a menor atividade de chaveamento. Esse exemplo também mostra que o compartilhamento de recursos no tempo pode modificar significativamente a característica dos sinais, causando um incremento na atividade de chaveamento e, conseqüentemente, na dissipação de potência.

Embora o exemplo mostrado em [CHA 95] (Figura 4.2) para circuitos contadores aponte para uma menor dissipação de potência na implementação de barramentos paralelos, este fato pode não se repetir para circuitos maiores. Em [KOM 98], por exemplo, pode-se observar um estudo comparativo de arquiteturas baseadas em barramentos e em multiplexadores para aplicação em microprocessadores. Neste trabalho é mostrado que com a implementação de um circuito integrado de teste, a arquitetura baseada em multiplexadores apresentou uma redução de dissipação de potência de cerca de 30%, comparada à arquitetura baseada em barramento. O maior consumo de potência observado para a implementação baseada em barramentos é atribuído ao comprimento das suas linhas.

Estes resultados mostram que a seleção ótima de arquiteturas baseadas em barramentos ou multiplexadores é uma considerável decisão em termos de dissipação de potência e que a redução de linhas de barramentos é um dos caminhos para a redução da potência global de circuitos integrados.

4.1.2 Métodos de Codificação em Barramentos

A maneira mais comum de comunicação de dados entre diferentes *chips* em um circuito integrado, sem a utilização de codificação, é a transmissão e recepção dos dados na sua representação original (geralmente em binário) sobre um conjunto de n fios, como mostra a Figura 4.3(a). A idéia de utilizar um esquema de codificação para a redução da atividade de chaveamento nas linhas do barramento, é mostrada na Figura 4.3(b), onde se observa que a mudança de representação dos dados pode ser obtida ao custo de um *hardware* adicional para codificação/decodificação dos dados. Neste caso, observa-se pela Figura 4.3(b) que há a possibilidade de utilização de um número diferente de linhas no barramento (m fios) entre as etapas de codificação/decodificação dos dados.

De acordo com [CHA 95], o número de linhas que é utilizado nos barramentos está relacionado com os tipos de técnicas aplicadas nas etapas de codificação/decodificação dos dados e podem ser classificadas em redundantes ou não redundantes. Um esquema de codificação é classificado como não redundante se $m = n$, e os 2^n elementos do conjunto de n-bit da palavra de dados são mapeados entre estas linhas. De outra forma, quando $m>n$, o esquema de codificação apresenta redundância, pois as 2^n únicas palavras de dados são mapeadas para um conjunto maior de 2^m palavras de dados para serem transmitidas pelas linhas do barramento.

O mapeamento dos dados a serem transmitidos pode ser realizado de diferentes formas, de acordo com o tipo de técnica que é empregada. A vantagem de cada técnica está associada ao tipo de mapeamento dos dados que é realizado, assim como à necessidade ou não de utilização de memória associada ao codificador/decodificador.

Diversas técnicas de codificação para redução de potência são apresentadas em [STA 97], onde é mostrado que cada técnica pode reduzir o consumo de potência

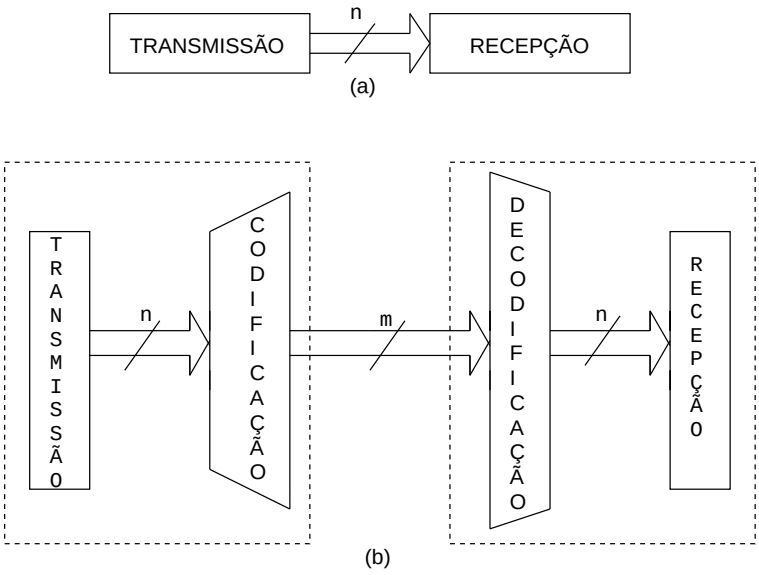

FIGURA 4.3 – Esquemas de transmissão e recepção de dados em barramento.

por informações transmitidas em linhas de barramentos de comunicação com elevados valores de capacitância chaveada. A redução de potência nos barramentos é obtida a partir da redução da atividade de transição dos dados.

4.1.2.1 Codificação *One-Hot*

Esta técnica utiliza um esquema de codificação redundante com o mapeamento de um para um entre os n-bits da palavra de dados a ser enviada e os m bits das palavras de dados que são transmitidas. A conexão entre os diferentes *chips* é feita utilizando-se $m = 2^n$ fios, e uma palavra de dados de n bits é codificada para transmissão, através da colocação do nível lógico 1 no i-ésimo fio, e nível lógico 0 nos m-1 fios restantes, onde $0 \leq i \leq 2^n$-1 é o valor binário correspondente ao padrão dos bits a transmitir.

Embora esta técnica possa proporcionar uma grande redução na atividade de chaveamento, a sua grande desvantagem se refere à quantidade de fios necessária que é proporcional a 2^n. De acordo com [CHA 92a], para n=8, esta técnica possibilita uma redução de 75% na atividade de chaveamento, entretanto são necessários m=256 fios para a codificação da palavra.

4.1.2.2 Codificação *Bus-Invert*

Neste método de codificação, a palavra de dados pode ser invertida e transmitida caso isto represente uma redução do número de transições [STA 95]. Neste método, uma linha de barramento extra, chamada *invert*, é utilizada para indicação da inversão ou não da palavra de dados.

O método calcula a distância de *Hamming* entre duas palavras consecutivas, ou seja, verifica-se a quantidade de bits diferentes no barramento composto por n linhas. Se a distância de *Hamming* entre a próxima palavra e a palavra atual transmitida é menor ou igual a $n/2$, a próxima palavra é transmitida na sua forma original, com a linha extra *invert* igual a 0. De outra forma, cada bit da próxima

palavra de dados é invertido e a linha extra *invert* é igual a 1, indicando que houve mudança de representação da palavra de dados transmitida.

Um exemplo de aplicação do código *bus-invert* pode ser visto na Figura 4.4, para uma palavra de dados de 8 bits com um bit extra de indicação de inversão da palavra. Na Figura 4.4, observa-se uma redução de quase 40% na atividade de transição, com a aplicação da codificação *bus-invert*, incluindo a linha extra de indicação de inversão da palavra de dados.

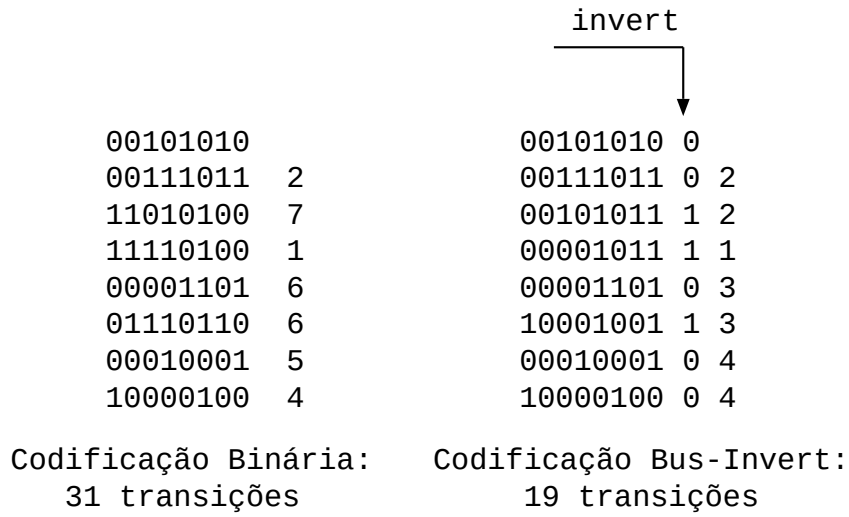

FIGURA 4.4 – Exemplo de aplicação do código *bus-invert*.

Apesar da técnica *bus-invert* ser de simples aplicação com consideráveis valores de redução de atividade de transição, o processamento da distância de *Hamming* entre palavras sucessivas necessita do uso de uma quantidade considerável de *hardware*. Desta forma, esta técnica se torna de difícil aplicação em grandes barramentos.

4.1.2.3 Codificação Gray

Um dos métodos de codificação mais utilizado para redução da atividade de transição em barramento de endereços é a utilização de código gray [MEH 96b]. Em uma seqüência de contagem em código gray apenas um bit muda de representação entre valores consecutivos, podendo produzir, desta forma, uma redução em até 50% no número de transições em relação ao código binário, como mostra o exemplo da Figura 4.5. Desta forma, a utilização deste código se torna importante para a redução da atividade de chaveamento quando os dados são seqüenciais ou apresentam um alto grau de correlação.

Embora o código gray apresente uma grande redução no número de transições e não necessite de nenhuma linha de barramento adicional para sua aplicação, deve-se observar que a conversão de representação entre este código e o código binário se torna complexa. A conversão do código binário para o código gray é efetuada de acordo com as Equações 4.3 e 4.4 [CHA 95], onde B=(b_{n-1},b_{n-2},$\cdots$,b_1,b_0) é a representação binária e G=(g_{n-1},g_{n-2},$\cdots$,g_1,g_0) é a representação em código gray do número.

$$g_{n-1} = b_{n-1} \tag{4.3}$$

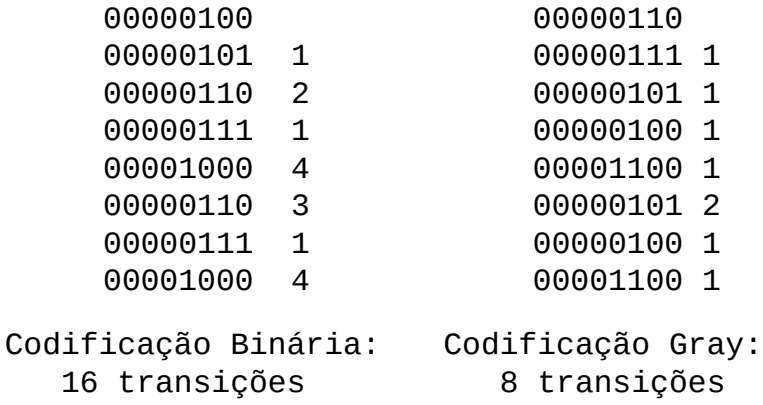

FIGURA 4.5 – Exemplo de aplicação do código gray.

$$g_i = b_{i+1} \oplus b_i \qquad (i = n-2, 0) \tag{4.4}$$

Como pode ser visto pelas Equações 4.3 e 4.4, a conversão do código binário em código gray consiste em repetir o bit mais significativo da palavra binária e utilizar operações lógicas EXOR entre todos os bits consecutivos da palavra. A conversão do código gray em código binário é mostrada nas Equações 4.5 e 4.6. Esta conversão também é realizada utilizando operações lógicas EXOR, entretanto de uma forma mais complexa. Esta maior complexidade está associada ao maior caminho crítico observado nesta conversão, pelo fato do bit a ser convertido depender do resultado da conversão anterior.

$$b_{n-1} = g_{n-1} \tag{4.5}$$

$$b_i = b_{i+1} \oplus g_i \qquad (i = n-2, 0) \tag{4.6}$$

O código gray tem sido aplicado em linhas de endereços de códigos para acessos de dados e instruções para redução da atividade de chaveamento. Em [SU 95] é observado que o uso de código gray em endereços de memórias *cache* pode reduzir o consumo de potência em até 30% nas *caches* de instrução.

4.1.2.4 Codificação *Beach*

Este método de codificação identifica a correlação de uma determinada seqüência de dados que é transmitida pelo barramento [BEN 97].

A estratégia é utilizada a partir de um *trace* de execução típico do barramento de endereços a ser codificado. Neste caso, este esquema de codificação identifica possíveis correlações em blocos de dados, a partir da estatística do sinal no barramento. As linhas do barramento são agrupadas de acordo com a correlação. Após a identificação dos grupos de dados com maior grau de correlação é gerada automaticamente uma função de codificação para cada grupo de bits e cada configuração de bits é transformada em uma nova representação. Os dados de saída são organizados de tal modo que o número médio de transições entre dados consecutivos é minimizado.

A Figura 4.6 mostra um exemplo da aplicação do código *beach* para uma palavra de dados de 8 bits. A seqüência de dados do lado esquerdo da Figura 4.6 apresenta a representação binária. Após a aplicação da estratégia de codificação *beach* esta seqüência de dados apresenta uma redução de 29% no número de transições.

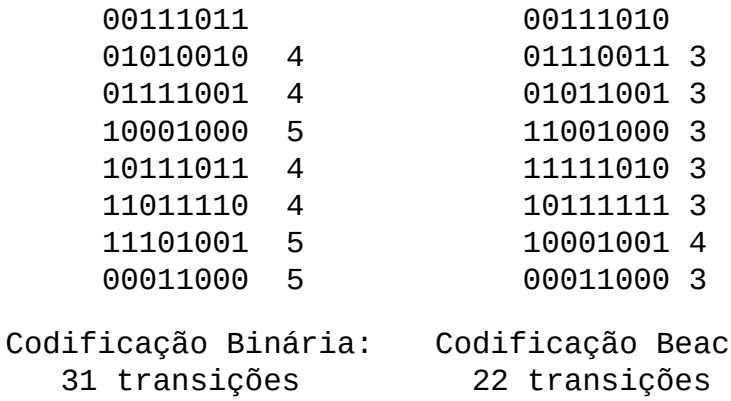

FIGURA 4.6 – Exemplo de aplicação do código *beach*.

No exemplo da Figura 4.6, é observado que na seqüência de dados na representação binária os três bits mais significativos apresentam o comportamento de um contador *up* e os dois bits menos significativos, desta mesma seqüência, apresentam o comportamento de um contador *down*. Desta forma, a estratégia de codificação *beach* consistiu em usar o código gray a estes dois grupos de bits no sentido de reduzir a atividade de transição da seqüência de dados.

Como mencionado, na técnica de codificação *beach* é gerada uma função de codificação, de acordo com a correlação existente nos blocos de dados agrupados. Desta forma, esta estratégia se torna interessante na redução da atividade de transição, pois existe um grau de liberdade na utilização da melhor função de codificação de acordo com a correlação nos grupos de dados. Por outro lado, observa-se que o fato de não existir um padrão que estabeleça sempre uma mesma estratégia a ser utilizada nos blocos de dados pode requerer constantes mudanças no projeto do sistema, visto que diferentes funções podem ser geradas para cada padrão de dados. Esta constante mudança no projeto do sistema pode causar uma sobretaxa em termos de área e linhas de barramento, limitando a utilização desta técnica a casos muito particulares.

4.1.2.5 Codificação *Transition*

A técnica de codificação *transition* é utilizada para transformar a representação do código do barramento antes que a operação aritmética seja realizada. As informações a serem codificadas ou decodificadas são comparadas com os dados anteriores que são armazenados. Desta forma, a utilização desta técnica garante que existe uma transição no barramento todas as vezes em que o dado a ser transmitido apresenta nível lógico 1 e não existe transição no barramento na situação contrária, como mostra o exemplo da Figura 4.7.

No exemplo da Figura 4.7, o lado esquerdo representa uma seqüência de palavras de dados de 8 bits em código binário. Após a codificação desta seqüência pela utilização da técnica *transition*, há uma redução de aproximadamente 21% na

```
01101001            01101001
00110110  6         01011111 4
10010110  2         10100000 8
10101001  6         00111111 6
01011110  7         11110111 3
00100101  6         01111011 3
11011110  7         11111011 1
11101011  4         00110101 5

Codificação Binária:  Codificação Transition:
   38 transições          30 transições
```

FIGURA 4.7 – Exemplo de aplicação do código *transition*.

atividade de transição. Esta redução na atividade de transição ocorre pelo fato de que cada nova palavra de dados que chega para ser transmitida pelo barramento é comparada bit a bit com a palavra de dados anterior codificada que fica armazenada em um registrador. Esta operação de comparação é realizada por uma porta lógica EXOR, conforme mostra o esquema da Figura 4.8 [RAM 99]. Observa-se pela Figura 4.8 que independentemente dos *drivers* do barramento que precisam ser utilizados em alguns casos, a implementação da técnica de codificação *transition* utiliza duas portas lógicas EXOR e um registrador em cada componente de leitura ou escrita do barramento.

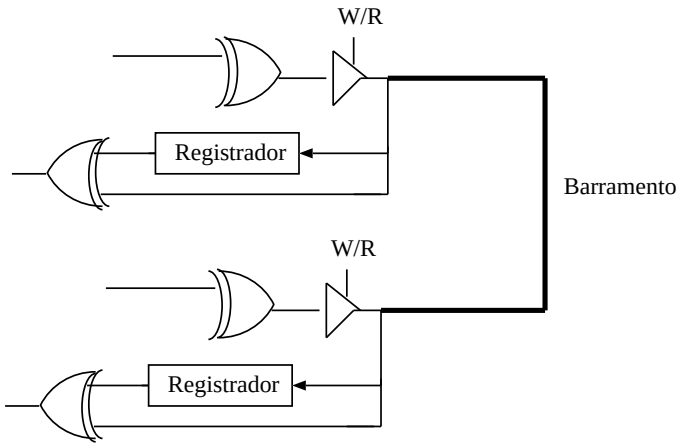

FIGURA 4.8 – Estrutura para implementação da técnica de codificação *transition*.

Em [RAM 99], observa-se que a utilização da técnica *transition* é eficiente na redução da atividade de transição, quando aplicada em barramentos de dados em configurações de memórias *cache* unificada e dual (dados e instruções). Comparado com a técnica *bus-invert*, a técnica *transition* apresentou um ganho médio em redução da atividade de transição de 18,6% no barramento de memória *cache* unificada para diferentes padrões de dados de entrada de *traces* (*SPEC95 - benchmark*). De acordo com [RAM 99], estes ganhos estão relacionados com o comportamento das palavras de dados utilizadas que alternam entre palavras com uma pequena quantidade de dados em nível lógico 1, com palavras de dados que apresentam uma distribuição mais uniforme de valores em níveis lógicos 0 e 1.

Apesar da técnica *transition* ter se mostrado eficiente em reduções da atividade de transição para barramento de dados nos exemplos com memórias *cache*, a sua utilização não apresentou nenhum ganho quando utilizada em barramentos de endereços destas memórias, mostrando que esta técnica não é recomendável para este tipo de aplicação.

4.1.2.6 Codificação *Bit Prediction*

Na técnica *bit prediction*, a idéia básica parte do princípio de que alguns barramentos exibem padrões de bits muito regulares. Por exemplo, em barramentos de endereços que exibem uma alta percentagem de endereços seqüenciais, pode ser utilizado um fator que possa fazer uma predição da próxima palavra de dados, com uma razoável precisão [RAM 99]. A predição do valor de um bit de dados pode ser realizada a partir da utilização de uma tabela com 2^k entradas, endereçadas pelos últimos k valores dos bits das palavras anteriores. Neste caso, se a predição é correta, não é realizada nenhuma transição no barramento. Entretanto, se a predição não é correta, a transição no barramento assinala que o valor na tabela está errado e que desta forma, este valor deve ser complementado e a tabela deve ser corrigida. A Figura 4.9 mostra a estrutura do *hardware* necessário para a implementação desta abordagem [RAM 99].

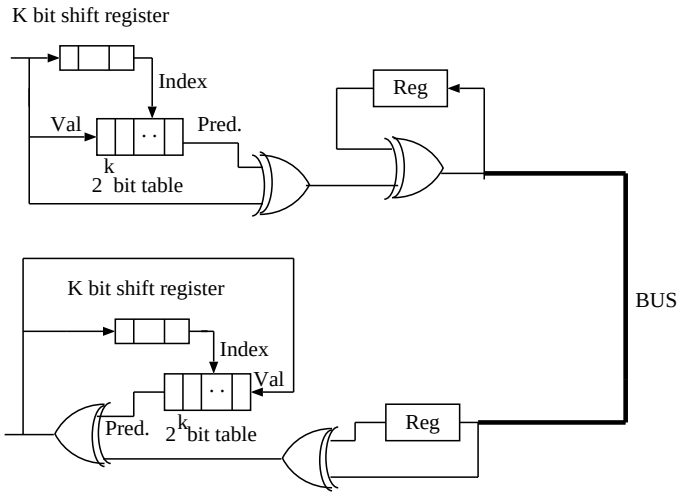

FIGURA 4.9 – Estrutura para implementação da técnica de codificação *bit prediction*.

Como pode ser observado na Figura 4.9, a técnica *bit prediction* apresenta a vantagem de não necessitar de linhas extras de barramento para sua implementação. Entretanto, esta técnica requer um *hardware* adicional comparada com a técnica de codificação *transition* mostrada na Figura 4.8.

Em [RAM 99], a aplicação da técnica *bit prediction* é comparada com as técnicas *transition* e *bus invert* para aplicação em barramentos de dados e endereços de memórias *cache* unificada e de dados e instruções. Observa-se que a técnica *bit prediction* apresenta valores de redução de atividade de transição de até 50% quando aplicada em barramentos de endereços de memórias *cache* de instrução, onde se tem uma maior seqüencialidade dos dados. Entretanto, esta técnica não se mostra eficiente quando aplicada em barramentos de dados, mostrando não ser indicada para

TABELA 4.1 – Exemplo de aplicação da técnica LWC.

Decimal	Binário	2-LWC	1-LWC
0	0000	00000	000000000000000
1	0001	00001	000000000000001
2	0010	00010	000000000000010
3	0011	00011	000000000000100
4	0100	00100	000000000001000
5	0101	00101	000000000010000
6	0110	00110	000000000100000
7	0111	11000	000000001000000
8	1000	01000	000000010000000
9	1001	01001	000000100000000
10	1010	01010	000001000000000
11	1011	10100	000010000000000
12	1100	01100	000100000000000
13	1101	10010	001000000000000
14	1110	10001	010000000000000
15	1111	10000	100000000000000

este tipo de aplicação. Além disto, deve-se observar que esta técnica apresenta uma estrutura de *hardware* que utiliza uma tabela com 2^k entradas para predição dos bits de dados, como mostra a Figura 4.9. Desta forma, a utilização desta técnica é conveniente somente para pequenos valores de k.

4.1.2.7 Codificação *Limited-Weight*

Este método de codificação é baseado na idéia de reduzir a probabilidade de ocorrência de valores lógicos 1 transmitidos no barramento. Isto é possível pela utilização de barramentos com maior quantidade de linhas e escolha de códigos que transportam muito menos valores lógicos 1 do que 0. Esta técnica foi proposta por [STA 97], e tem por idéia básica a observação de que a utilização de codificação baseada em transições, ao invés de codificação baseada em níveis, pode limitar o número de transições nas linhas de entrada. De acordo com [STA 97], o termo peso (*weight*) de uma palavra codificada representa o número de bits iguais a 1 nesta palavra.

Por definição, se o número de bits de uma palavra de código é n, então a aplicação da técnica *m-Limited Weight Coding - (mLWC)* pode resultar em palavras de código com peso $\leq m$, onde m representa o número máximo de bits em nível lógico 1 em uma palavra de dados. A Tabela 4.1 mostra um exemplo de aplicação da técnica *limited-weight* para valores de m iguais a 2 e 1 para uma seqüência de números n=4 bits [STA 97].

Observa-se pela Tabela 4.1 que enquanto a aplicação da técnica 1-LWC resulta em palavras com no máximo um bit em nível lógico 1, a aplicação da técnica 2-LWC resulta em palavras codificadas com no máximo 2 bits em nível lógico 1. Em ambos os casos, um maior número de linhas de barramento se faz necessário para a codificação das palavras de dados. Neste caso, como pode ser observado na Tabela 4.1, existe um relacionamento entre a menor quantidade de bits em nível

lógico 1 com a maior quantidade de linhas de barramento necessária para a codificação da palavra de dados. Deve-se observar que a técnica 1-LWC apresenta a mesma representação da codificação *one-hot*. De acordo com [STA 97a], a técnica 1-LWC (mínimo possível) tem muito pouca chance de ser utilizada na prática, exceto para valores muito pequenos de n, devido ao crescimento exponencial do número de linhas de barramento necessário. Entretanto, para o caso em que há um equilíbrio do valor de m aplicado para a codificação da palavra em relação ao número de bits da palavra original, esta técnica pode se tornar eficiente na redução da atividade de transição. Em [STA 97a], esta técnica foi aplicada a um sistema que possui um barramento com 9 bits de dados. Para este sistema, foi observado que a melhor estratégia de aplicação da técnica LWC consistiu em utilizar um valor de m=4. Para esta aplicação, houve uma redução de dissipação de potência de 50% com alguma correlação entre os dados. Entretanto, para o caso onde os dados apresentaram uma distribuição uniforme, houve uma redução de potência de aproximadamente 18%.

4.1.2.8 Complemento de 2 versus Sinal-Magnitude

Para a representação de números com sinal, utiliza-se tipicamente a representação de complemento de 2 para a representação de números. Nesta representação, observa-se que em uma palavra binária a região de dados é representada pelos bits menos significativos e a região de sinal é representada pelos bits mais significativos. A atividade de transição nestas regiões depende da operação que é efetuada, tendo-se neste caso a avaliação da correlação temporal e probabilidade de transição para os bits [LAN 95, RAM 97].

Um problema relacionado à representação de complemento de 2 é a extensão do sinal que causa um chaveamento nos bits mais significativos quando ocorre uma transição de sinal (de positivo para negativo ou vice-versa). Visto que uma grande parte dos bits mais significativos são utilizados para realizar a extensão do sinal, pode-se ter uma elevada atividade de transição quando o número é representado por sinal. Desta forma, torna-se interessante reduzir a faixa dinâmica da região de sinal com sua representação sendo feita pelo menor número de bits possível.

Na representação do número em sinal-magnitude, apenas um bit é reservado para indicação do sinal da operação e os outros bits são reservados para a magnitude dos dados. Desta forma, a atividade de transição relacionada à região de sinal é bem menor com apenas um bit sendo chaveado em operações de números com representação de sinal, como mostrado em [CHA 95]. Como conseqüência, tem-se uma menor dissipação de potência em relação à representação em complemento de 2.

4.1.3 Ordenamento dos Sinais de Entrada

O objetivo do ordenamento de sinais de entrada é reduzir a atividade de chaveamento de um circuito. Isto pode ser efetuado a partir de operações de associatividade e comutatividade aplicadas aos sinais de entrada dos circuitos. A Figura 4.10 mostra as duas operações para um exemplo em que uma operação de multiplicação é decomposta em operações de deslocamento e soma [CHA 95].

As operações de deslocamento na Figura 4.10 representam uma operação de *scaling* que tem o efeito de reduzir a faixa dinâmica do sinal. Na Figura 4.10, tem-se probabilidades de transição para o sinal *IN* e para as operações de deslocamento (*IN*

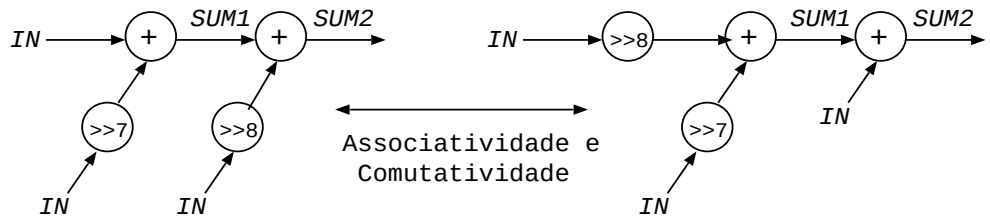

FIGURA 4.10 – Redução da atividade de chaveamento pela reordenação das entradas.

$\gg 7$ e $IN \gg 8$) para uma palavra de 16 bits, tendo-se o sinal IN com uma faixa dinâmica elevada (probabilidade alta para quase todos os bits), e as operações de deslocamento com uma faixa dinâmica mais baixa.

Observa-se pela Figura 4.10 que na primeira arquitetura a atividade de chaveamento é elevada, pois para o primeiro somador, tem-se a soma do sinal IN de elevada faixa dinâmica com a operação de deslocamento ($IN \gg 7$) com baixa faixa dinâmica, tendo-se desta forma, a soma *SUM1* resultante da primeira operação, com elevada faixa dinâmica. A segunda operação de soma é efetuada entre *SUM1* e a operação de deslocamento de baixa faixa dinâmica ($IN \gg 8$), tendo-se como resultado, a soma *SUM2* de elevada faixa dinâmica.

Para a segunda arquitetura na Figura 4.10, observa-se a reordenação das entradas após as operações de associatividade e comutatividade. Para este caso, a primeira soma resultante *SUM1* será de baixa faixa dinâmica, pois ocorre entre duas operações de deslocamento. A segunda soma terá uma maior faixa dinâmica, pois o sinal IN de alta faixa dinâmica é incluído na operação.

Observa-se então, que a segunda arquitetura, após o reordenamento das entradas, proporciona uma maior redução da atividade de chaveamento. De acordo com [CHA 95], na segunda arquitetura tem-se 30% menos capacitância chaveada em relação à primeira arquitetura.

4.1.4 Redução da Atividade de *Glitching*

Como visto anteriormente, o projeto de circuitos integrados estáticos pode apresentar transições espúrias, devido à propagação de atrasos de um bloco lógico a outro. Desta forma, um nó pode apresentar múltiplas transições em um período de relógio antes de alcançar o seu nível lógico correto.

Para minimizar as transições extras e, conseqüentemente, a dissipação de potência, torna-se importante o balanceamento de todos os caminhos de sinais e a redução da profundidade lógica [FAV 95, CHA 95, RAG 99].

A redução da atividade de *glitching* é mostrada no exemplo da Figura 4.11 [CHA 95] para a representação da operação de soma de 4 números, tendo-se uma implementação em cascata (implementação não *pipeline*) e outra em árvore. No exemplo, assume-se que todas as entradas primárias estão disponíveis ao mesmo tempo.

Observa-se pela Figura 4.11 que na primeira implementação existe a propagação de atraso através do primeiro somador. Neste caso, quando a entrada C fica disponível, o segundo somador efetua a operação com o valor anterior do primeiro somador. Após a propagação do atraso, o valor correto do primeiro somador fica

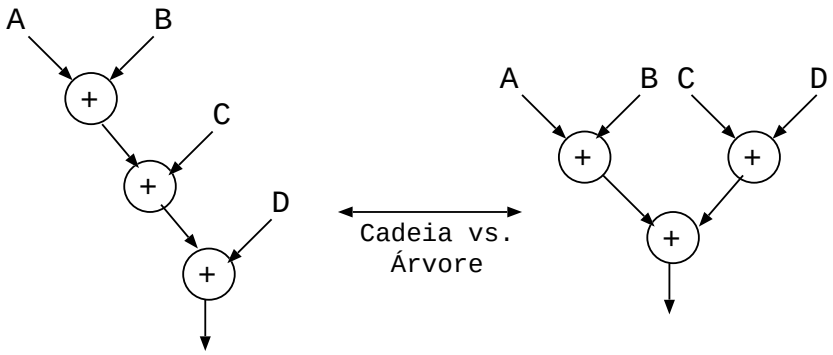

FIGURA 4.11 – Representação de operações para redução da atividade de *glitching*.

disponível e o segundo somador efetua novamente a operação, tendo-se duas operações por ciclo de relógio. Da mesma forma, ocorre a propagação de atrasos através do segundo somador e o terceiro somador efetua 3 operações por ciclo.

Para a segunda implementação da Figura 4.11 (implementação em árvore), os caminhos dos sinais são mais balanceados e existe uma menor possibilidade de transições extras.

De acordo com [CHA 95], a capacitância chaveada para uma implementação em cascata é um fator de 1,5 maior em relação à implementação em árvore para a operação de adição de 4 entradas e 2,5 maior para a adição de 8 entradas.

A técnica de otimização de potência através da análise de *glitching* pode ser vista em [RAG 96], onde para um circuito que realiza o maior divisor comum de dois números, são empregados métodos que incluem a reestruturação da rede de multiplexadores (para produção de correlação de dados, eliminação de *glitches* em sinais de controle e redução de *glitches* em sinais de dados) e inclusão de atrasos seletivos de subida e descida. A inclusão destes atrasos ocorre onde existe uma probabilidade alta de ocorrência de transições nos blocos do circuito.

4.2 Redução de Potência Utilizando Transformações

A aplicação de transformações deve atuar na estrutura computacional de um circuito, de forma que o seu comportamento de entrada/saída seja preservado. A aplicação de transformações deve explorar um número de alternativas arquiteturais e selecionar as que resultam em menor consumo de potência, de acordo com o número e tipo de módulos computacionais, suas interconexões e a seqüência de operações [CHA 92a, MEH 94].

De acordo com [CHA 95], existem dois caminhos de otimização de potência usando transformações. O primeiro caminho consiste na redução da fonte de alimentação através da aplicação de transformações em velocidade (transformações normalmente usadas para otimização de desempenho). O segundo caminho envolve a minimização da capacitância efetiva através da aplicação de transformações mais genéricas voltadas para o caminho crítico e que envolvem, por exemplo, a redução do número de operações necessárias, a substituição de algumas operações por outras mais convenientes ou eficientes, a otimização da utilização de recursos e a redução do comprimento da palavra para representação dos dados.

4.2.1 Transformações em Velocidade

Segundo [CHA 95], a idéia básica desta técnica é reduzir o número de passos de controle, de forma que os ciclos de relógio de controle mais lentos possam ser utilizados para um valor de desempenho fixo permitindo uma redução na fonte de alimentação. A redução nos requerimentos de passos de controle é muitas vezes possível devido à exploração de concorrência que inclui *retiming/pipelining*, transformações algébricas e transformações *loop unrolling*.

4.2.1.1 *Retiming*

A operação de *retiming* consiste em reposicionar os registradores nos circuitos seqüenciais, mantendo-se o seu comportamento funcional. A operação de *retiming* foi proposta inicialmente por [LEI 83] como uma técnica para melhorar o desempenho pela movimentação dos registradores ao longo dos circuitos. Enquanto que na Seção 3.4 foi discutido o uso de *retiming* para redução da atividade de chaveamento, nesta parte do trabalho é abordada a sua utilização para a redução do caminho crítico com redução da tensão de alimentação. A Figura 4.12 mostra a aplicação de *retiming* a um filtro de resposta ao impulso infinito (*Infinite Impulse Response - IIR*) de segunda ordem [CHA 92, CHA 95, CHA 95a]. O circuito da Figura 4.12 apresenta, de acordo com [CHA 95], uma energia gasta por período de amostragem igual a 10,28nJ. As Figuras 4.13(a) e 4.13(b) mostram a exploração arquitetural para o mesmo circuito após a aplicação de duas operações de *retiming*.

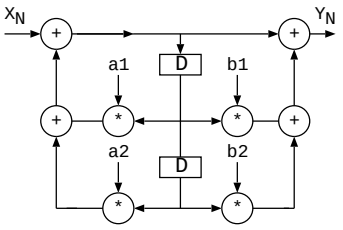

FIGURA 4.12 – Filtro IIR de segunda ordem.

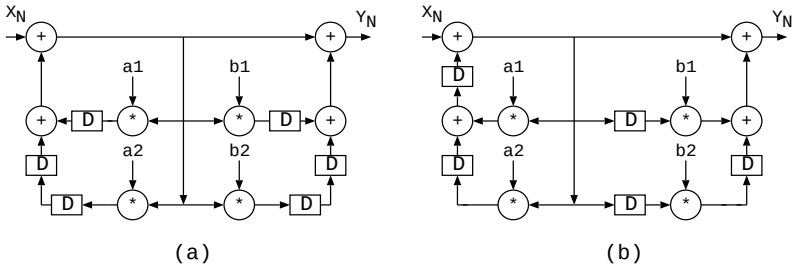

FIGURA 4.13 – Filtro IIR de segunda ordem após aplicação de *retimings*.

Observa-se pela Figura 4.13, que em ambas as implementações a movimentação dos registradores é realizada no sentido de preservar o mesmo desempenho dos circuitos. Embora ambas as implementações da Figura 4.13 apresentem mesmos valores de caminhos críticos (igual a 3), a implementação da Figura 4.13(b) se torna mais

conveniente, por apresentar menos registradores. De acordo com [CHA 95], a implementação da Figura 4.13(b) apresenta um consumo de energia de 9,85nJ, o que representa uma redução de 17,4% em relação à implementação inicial da Figura 4.12. De outra forma, a implementação da Figura 4.13(a) apresenta um consumo de energia por período de amostragem igual a 11,92nJ, valor superior em relação às duas outras implementações.

4.2.1.2 Processamento Paralelo e *Pipelining*

As transformações empregadas para a redução de tensão de acordo com a freqüência de operação dos circuitos são baseadas no aumento do nível de concorrência no sistema, ou seja, mais *hardware* pode ser utilizado e diversas tarefas podem ser realizadas. Neste caso, as transformações mais típicas são *pipeline* e paralela [POT 92, GOO 94, CHA 95, MEH 98]. Contudo, operações em paralelo e *pipeline* resultam em penalidade de área, pois requerem *hardware* adicional. Um aumento em área pode resultar em aumento em capacitância chaveada, com aumento em dissipação de potência. Entretanto, a potência diminui quadraticamente com a tensão de alimentação (V_{dd}) e aumenta somente linearmente com a capacitância chaveada. Deste modo, esta técnica pode ser empregada com bons resultados.

No caso do processamento paralelo, o *hardware* é replicado de tal forma que múltiplas computações podem ser realizadas simultaneamente, mantendo-se o desempenho, possibilitando neste caso uma redução da fonte de alimentação. Para o caso em que dois circuitos idênticos são utilizados, tem-se cada unidade trabalhando na metade da sua taxa original mantendo o desempenho original. O exemplo da Figura 4.14 mostra a utilização das técnicas paralela e *pipeline* a partir da arquitetura de um circuito somador-comparador [CHA 95].

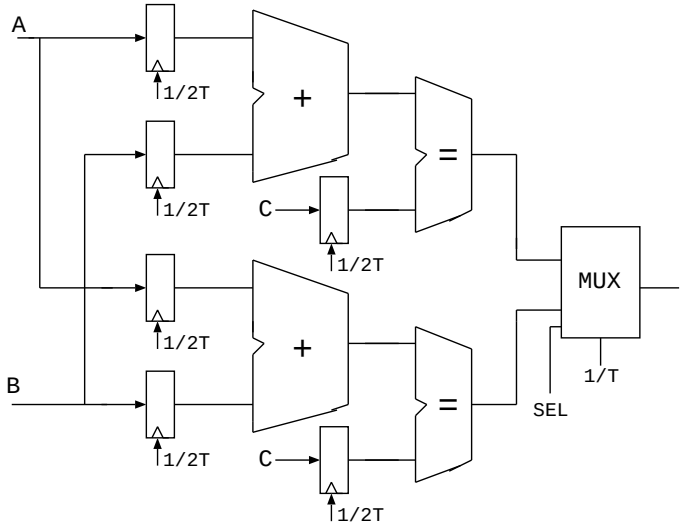

FIGURA 4.14 – Arquitetura paralela para o circuito somador-comparador exemplo.

Como mostra a Figura 4.14, dois *datapaths* idênticos são utilizados permitindo cada unidade trabalhar na metade da sua taxa original mantendo o seu desempenho original. Para o caso da tensão de alimentação utilizada ter um valor igual a 5V, pode-se ter a sua redução de 5V para 2,9 V (tensão em que o atraso dobra, como mostrado em [CHA 95] para circuitos somadores, multiplicadores, osciladores, etc).

Deste modo, enquanto a freqüência de operação tem um fator de redução de 2, a capacitância apresenta um fator de acréscimo de 2, devido à utilização do dobro dos *datapaths*. Além disto, tem-se um acréscimo no valor de capacitância devido a roteamento extra, tendo-se um fator de incremento total de 2,15 extraído de *layout*, como mostrado em [CHA 95]. Logo, a potência do *datapath* paralelo é dado de acordo com a Equação 4.7.

$$P_{par} = C_{par} V_{par}^2 f_{par} = (2,15 C_{ref})(0,58 V_{ref})^2 \frac{f_{ref}}{2} \approx 0,36 C_{ref} V_{ref}^2 f_{ref} \qquad (4.7)$$

Outro caminho para redução de potência, com redução da fonte de alimentação, é a utilização da técnica *pipeline*. Esta técnica é utilizada para obter implementações de circuitos, em uma maior velocidade, ao custo também de aumento do *hardware*, como mostra o exemplo da Figura 4.15, para o mesmo circuito somador-comparador.

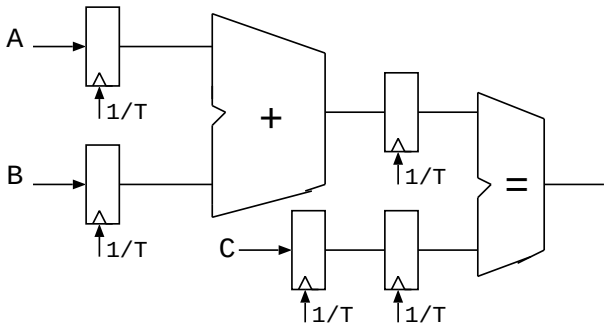

FIGURA 4.15 – Arquitetura *pipeline* para o circuito somador-comparador exemplo.

Como pode ser observado pela Figura 4.15, com o registrador adicional, o caminho crítico se torna $\max(T_{somador}, T_{comparador})$, permitindo ao somador e ao comparador operarem em uma taxa mais lenta. Para este exemplo, novamente os dois atrasos são iguais, permitindo que a fonte de alimentação seja reduzida de 5V para 2,9V (caso esta tensão de alimentação seja utilizada), mantendo-se o desempenho. Observa-se que existe para este caso, um pequeno aumento em *hardware*, devido à adição de registradores extras. Entretanto, este aumento é inferior ao observado na arquitetura paralela. Neste exemplo, há um aumento na capacitância efetiva por um fator de aproximadamente 1,15 [CHA 95]. Logo, a potência consumida pelo circuito utilizando a arquitetura *pipeline* é dada pela Equação 4.8.

$$P_{pipe} = C_{pipe} V_{pipe}^2 f_{pipe} = (1,15 C_{ref})(0,58 V_{ref})^2 f_{ref} \approx 0,39 C_{ref} V_{ref}^2 f_{ref} \qquad (4.8)$$

Uma extensão da análise pode ser realizada a partir da utilização da combinação das técnicas *pipeline* e paralela. De acordo com [CHA 95], para este tipo de arquitetura, há uma redução do caminho crítico e desta forma dos requerimentos de velocidade por um fator de 4. Logo, a tensão pode ser reduzida para 2V, onde o valor de atraso aumenta por um fator de 4 e o consumo de potência, para este caso, é dado de acordo com a Equação 4.9.

$$P_{parpipe} = C_{parpipe} V_{parpipe}^2 f_{parpipe} = (2,5 C_{ref})(0,4 V_{ref})^2 \frac{f_{ref}}{2} \approx 0,2 C_{ref} V_{ref}^2 f_{ref} \qquad (4.9)$$

A Tabela 4.2 mostra os resultados de área e potência normalizados para diferentes estilos de arquiteturas baseadas em escalonamento da tensão de alimentação para a parte operativa do circuito somador-comparador [CHA 95].

TABELA 4.2 – Resultados de arquiteturas com escalonamento de V_{dd}.

Arquiteturas	Tensão	Área (Normalizada)	Potência (Normalizada)
Normal	5V	1	1
Paralela	2,9V	3,4	0,36
Pipeline	2,9V	1,3	0,39
Pipeline-Paralela	2V	3,7	0,2

Observa-se na Tabela 4.2 que com a arquitetura *pipeline*, há uma redução de potência praticamente da mesma ordem da arquitetura paralela, com a vantagem da utilização de menor área. A implementação paralela-*pipeline* resulta em uma redução de potência da ordem de 5 vezes, tendo-se entretanto para esta implementação, a maior área utilizada.

4.2.1.3 Aplicação de *Loop Unrolling*

Loop unrolling é uma técnica de paralelização que é amplamente utilizada em programas que gastam a maior parte do tempo de processamento em seus laços [KOS 97, HUA 99]. Desta forma, esta técnica pode ser utilizada para reutilização de dados o máximo possível, assim como paralelização de instruções.

A técnica *loop unrolling* é proposta como uma técnica de transformação que aplicada à arquitetura de circuitos, pode habilitar a utilização de outras técnicas que produzam impacto direto na redução do consumo de potência [HUA 94, CHA 95, WAN 95].

A Figura 4.16 mostra o efeito da aplicação da técnica de *loop unrolling* na estrutura de um filtro IIR de primeira ordem, onde duas amostras de saída são computadas em paralelo baseadas em duas amostras de entrada [CHA 95]. As grandezas estão especificadas em unidades arbitrárias, indicando análise comparativa apenas.

Pela estrutura do filtro da Figura 4.16(b), observa-se que o caminho crítico dobra em relação à estrutura original da Figura 4.16(a). Entretanto, como duas amostras são processadas em paralelo, o caminho crítico efetivo não muda e a tensão de alimentação não pode ser alterada. O valor da capacitância chaveada também não muda, pois o número de operações dobra, mas para processamento de duas amostras de entrada em paralelo. Como a capacitância chaveada e a tensão de alimentação não mudam, a potência desta implementação permanece inalterada, para um modelo de atraso zero, mostrando que a técnica de *loop unrolling* não afeta o consumo de potência nestas condições. Entretanto, esta técnica habilita outras transformações (distributividade, propagação de constantes e *pipelining*) que resultam em uma significativa redução no consumo de potência.

- **Aplicação de Transformação Algébrica e Propagação de Constantes**

A Figura 4.17 mostra o efeito da aplicação de transformação algébrica e propagação de constantes [HUA 94, POT 94, PAR 95, CHA 95] à estrutura do filtro IIR modificada da Figura 4.16(b).

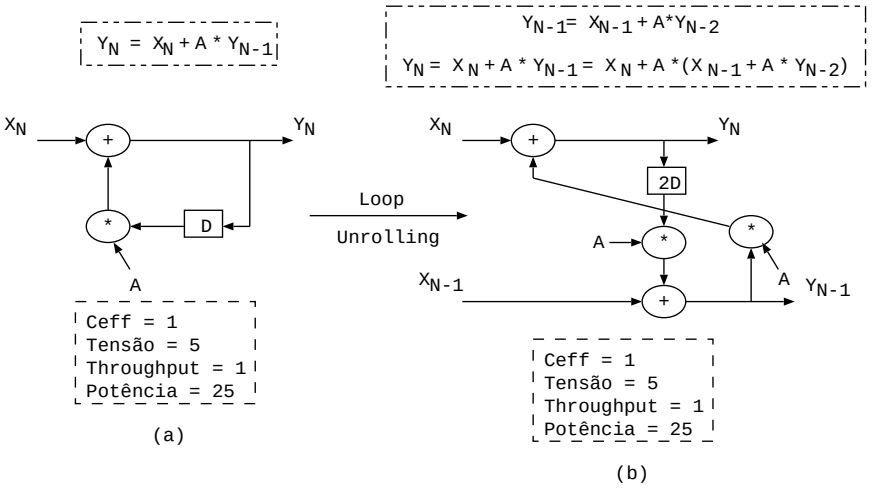

FIGURA 4.16 – Aplicação de *loop unrolling* na estrutura de um filtro IIR de primeira ordem.

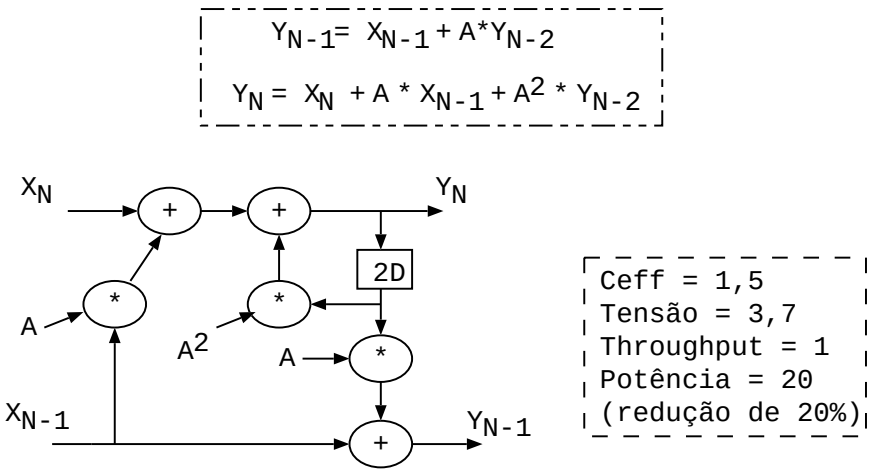

FIGURA 4.17 – Aplicação de transformação algébrica e propagação de constantes ao filtro IIR de primeira ordem.

Pela estrutura mostrada na Figura 4.17, observa-se um caminho crítico de valor igual a 3 para o processamento de duas amostras. Desta forma, o caminho crítico efetivo é reduzido e a tensão de alimentação pode também ser reduzida, resultando em uma redução de potência de 20% de acordo com [CHA 95].

- **Aplicação de *Pipelining***

A Figura 4.18 mostra a aplicação de *pipelining* à estrutura da Figura 4.17 para redução do caminho crítico [CHU 94, CHA 95].

Como mostrado na Figura 4.18, a aplicação de *pipelining* à estrutura da Figura 4.17 resulta na redução do caminho crítico para 2 ciclos. Como a estrutura transformada trabalha na metade da taxa de amostragem original (tem-se duas amostras sendo processadas em paralelo), e o caminho crítico é o mesmo do *datapath* original (2 ciclos de controle), a tensão de alimentação pode ser reduzida a 2,9V (onde o atraso dobra, de acordo com os exemplos mostrados em [CHA 95]).

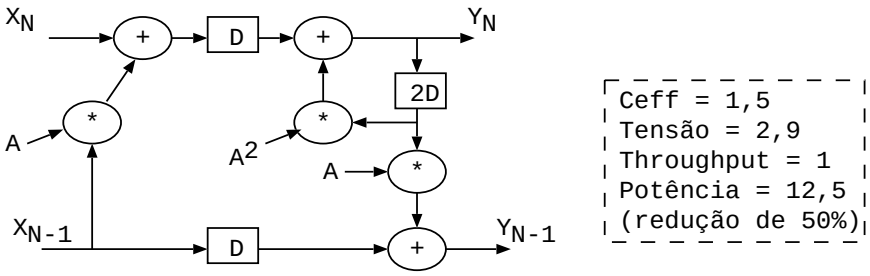

FIGURA 4.18 – Aplicação de *pipelining* ao filtro IIR de primeira ordem.

Observa-se pelos valores mostrados na Figura 4.18, o acréscimo da capacitância efetiva, visto que a estrutura transformada necessita de 3 multiplicações e 3 adições para o processamento de duas amostras, enquanto a estrutura original necessita de uma multiplicação e uma adição para processar uma amostra. A redução na fonte de alimentação compensa este acréscimo de 50% em capacitância, tendo-se desta forma, uma redução global de potência por um fator de 2, devido ao efeito quadrático da tensão de alimentação na potência.

4.2.2 Transformações Genéricas

Como visto no item anterior, um conjunto de transformações na estrutura do algoritmo pode levar a condições no projeto dos circuitos integrados que tenham impacto direto em velocidade e potência. Quando o alvo das otimizações desloca-se de velocidade para capacitância efetiva, utiliza-se um diferente conjunto de transformações algorítmicas, que possam contribuir diretamente para sua redução. Cada uma destas transformações visam principalmente, uma melhor utilização dos recursos disponíveis no sistema, uma vez que a utilização de uma menor quantidade de elementos de computação normalmente resultam em um melhor desempenho do projeto desenvolvido.

4.2.2.1 Redução do Número de Operações

O caminho mais efetivo para redução da capacitância chaveada é a redução do número de operações (e desta forma, o número de eventos de chaveamento) no grafo do fluxo de controle de dados. Deve-se observar entretanto, que a redução na capacitância de chaveamento não implica na redução do caminho crítico. Esta abordagem pode ser analisada a partir de dois exemplos utilizados em [CHA 95], mostrando o impacto da redução das operações na atividade de chaveamento. No primeiro exemplo, aplicam-se transformações em um polinômio de segunda ordem ($X^2 + AX + B$) e no segundo exemplo, aplicam-se transformações a um polinômio de terceira ordem ($X^3 + AX^2 + BX + C$).

- **Redução da capacitância com manutenção do desempenho**

A Figura 4.19(a) mostra a estrutura de implementação do polinômio de segundo grau, que necessita de duas operações de multiplicação e duas operações de adição e tem um caminho crítico igual a 3. Neste exemplo, assume-se que cada operação é realizada em um ciclo de controle.

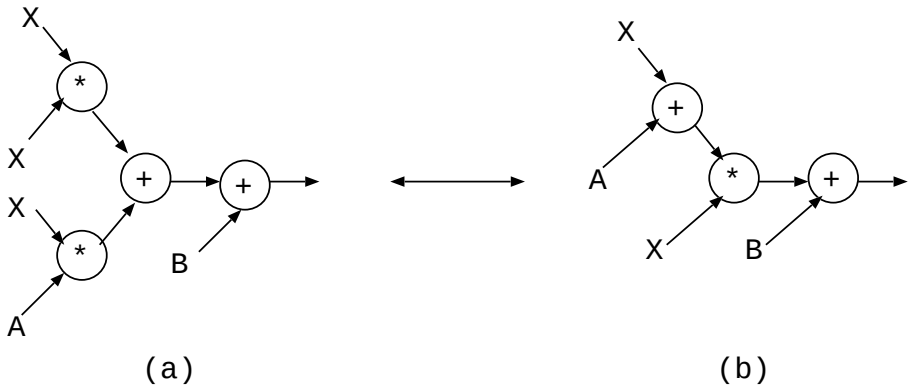

FIGURA 4.19 – Redução da capacitância chaveada com manutenção do desempenho.

Na Figura 4.19(b) é apresentada a implementação do polinômio de segundo grau com uma versão transformada, tendo-se uma diferente estrutura computacional. Observa-se que a estrutura transformada apresenta o mesmo caminho crítico da solução inicial da Figura 4.19(a), e desta forma, ambas as soluções apresentam o mesmo desempenho em uma determinada tensão de alimentação. Observa-se também, que a solução transformada (Figura 4.19(b)) apresenta uma operação de soma a menos, implicando em uma menor quantidade de capacitâncias chaveadas e dissipação de potência.

- **Redução da capacitância com aumento do caminho crítico**

A Figura 4.20(a) mostra a implementação do polinômio de terceiro grau, onde se observa uma significativa redução no número de operações, tendo-se entretanto um maior caminho crítico com um valor igual a 4.

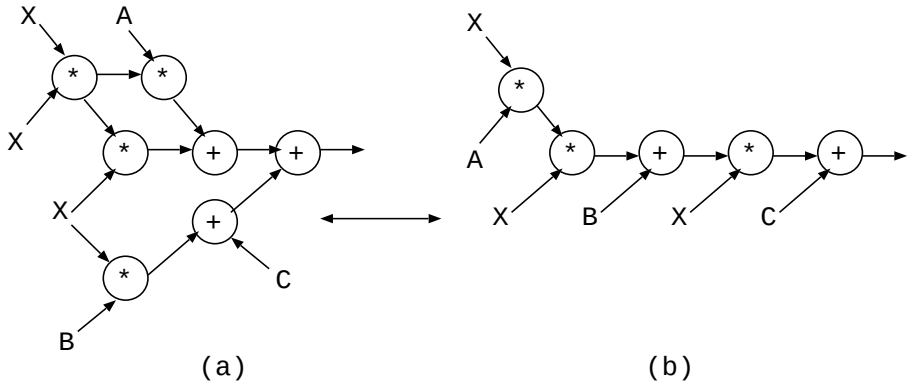

FIGURA 4.20 – Redução de capacitância com aumento do caminho crítico.

Observa-se pela Figura 4.20(b) que pela aplicação da transformação algébrica no polinômio de terceiro grau, ocorre a redução do número de multiplicações, tendo-se desta forma, uma redução da capacitância efetiva. Entretanto, observa-se que o caminho crítico sofre um aumento, tendo nesta configuração, um valor igual a 5. Neste caso, observa-se a necessidade de aumento na fonte de alimentação em relação à implementação inicial da Figura 4.20(a), para manutenção do desempenho. Os

exemplos acima mostram que a aplicação de transformações na estrutura computacional de circuitos pode ter diferentes efeitos na capacitância chaveada e tensão de alimentação. De acordo com [CHA 95], as transformações que reduzem diretamente o número de operações em um CDFG incluem eliminação de subexpressões comuns e distributividade.

4.2.2.2 Substituição de Operações

A técnica de transformações utilizando a substituição de operações parte do princípio de que algumas operações necessitam de menos energia por computação, podendo ser utilizadas para substituição de outras operações.

A idéia inicial de exemplo que considera a substituição de operações para redução da capacitância chaveada é a utilização de operações de adição em substituição à operações de multiplicação. Entretanto, [CHA 95] mostra que este tipo de transformação pode muitas vezes aumentar o caminho crítico, como mostra o exemplo da Figura 4.21.

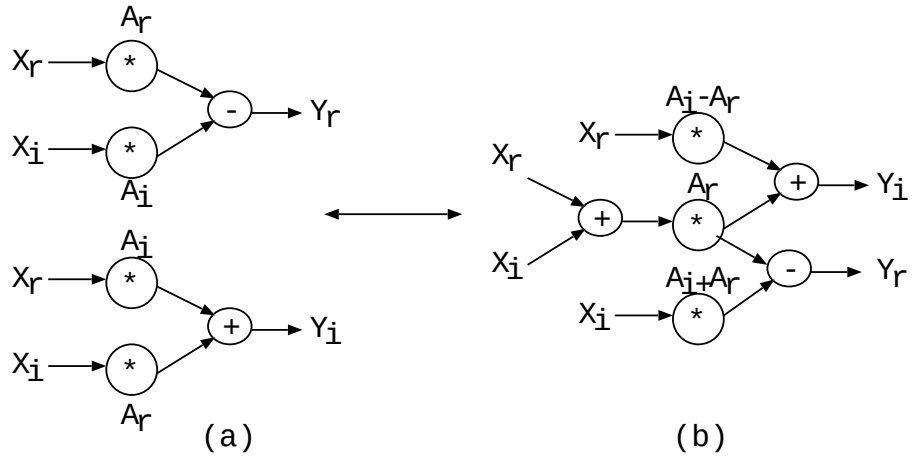

FIGURA 4.21 – Substituição de operação de multiplicação por adição.

Nos dois CDFGs mostrados na Figura 4.21, ambos os sistemas realizam a multiplicação de um número complexo X por uma constante complexa A. A implementação mostrada na Figura 4.21(a) é realizada através de um circuito somador, um circuito subtrator e quatro circuito multiplicadores e apresenta um caminho crítico de valor igual a 2. No CDFG modificado mostrado na Figura 4.21(b), observa-se a aplicação de algumas transformações algébricas, tendo-se um multiplicador substituído por um somador. Esta operação a princípio parece interessante para otimização de consumo de potência. Entretanto, o caminho crítico aumenta para um valor igual a 3 e, desta forma, é permitido um menor valor de redução na tensão de alimentação.

Outra importante transformação neste nível é a substituição de operações de multiplicação com valores constantes por uma combinação de operações de deslocamento e soma. Esta aplicabilidade é importante em circuitos DSP, onde são comuns operações de multiplicações, como será visto posteriormente [NGU 2000].

4.2.2.3 Redução do Comprimento da Palavra

O número de bits usado em uma operação afeta todos os parâmetros de um projeto que incluem velocidade, área e potência. De acordo com [CHA 95], existem 3 principais razões para minimização do número de bits durante otimização de potência: (1) menos eventos de chaveamento e menos capacitâncias chaveadas, (2) a operação funcional pode ser feita mais rapidamente e desta forma, a tensão de alimentação pode ser reduzida, mantendo o desempenho constante, (3) uma menor quantidade de bits não somente reduz o número de linhas de transferência, mas também reduz o comprimento de interconexões médias e capacitâncias.

Em [CHA 95] é mostrado um exemplo de implementação de um filtro *Avenhaus* passa faixa de ordem 80, onde foi realizada uma comparação entre duas diferentes implementações. Enquanto na primeira implementação, o filtro utilizou uma palavra de 23 bits para obtenção de estabilidade numérica, na segunda implementação (paralela) foi utilizada uma palavra de 11 bits. De acordo com [CHA 95], a segunda implementação apresentou uma redução de dissipação de potência em um quarto do valor obtido na primeira implementação. Outro aspecto destacado em [CHA 95] é a redução do caminho crítico, que apresentou na segunda implementação um valor cerca de 33% menor.

4.3 Resumo

Neste capítulo foram apresentadas técnicas para redução do consumo de potência no nível arquitetural. A aplicação de diferentes esquemas de codificação em barramentos de dados e endereços é uma das principais técnicas de redução do consumo de potência neste nível. Neste contexto, foram apresentados diferentes esquemas de codificação para redução da atividade de transição nos barramentos. Como pôde ser observado, embora a maioria destas técnicas consigam reduções significativas de potência, necessitam de linhas de barramento adicionais para sua aplicação. No nível arquitetural, a utilização de técnicas de transformação também apresenta impacto na redução de potência de diferentes arquiteturas de circuitos. Neste contexto, a liberdade do projetista está na utilização de paralelização e seqüencialização de operações com impacto na quantidade de *hardware* que pode ser compartilhado. Esta liberdade pode ser usada para a exploração de arquiteturas que possibilitem a redução nos níveis de tensão da fonte de alimentação com manutenção do mesmo desempenho. Como foi abordado, algumas técnicas têm sido utilizadas com esta finalidade, tais como *retiming*, *pipelining* e *loop unrolling*. Algumas outras técnicas apresentadas visam a redução do número de operações para redução da capacitância chaveada. Estas técnicas partem do princípio de que algumas operações necessitam de menos energia por computação podendo, desta forma, serem utilizadas para substituição de outras operações.

5 Técnicas Desenvolvidas para a Redução de Potência

Como mostrado no Capítulo 4, uma das principais técnicas de redução de potência no nível arquitetural envolve a redução da atividade de chaveamento. Neste trabalho de pesquisa, as técnicas de redução de potência propostas têm por objetivo a redução da atividade de chaveamento dos circuitos, considerando os aspectos de correlação de sinais. Como a atividade de chaveamento é dependente da correlação entre sucessivos dados de entrada, o aumento da correlação resulta em redução do consumo de potência.

A codificação de dados é uma das técnicas mais utilizadas para redução da atividade de chaveamento nos barramentos de dados e endereços [SAC 98]. Neste trabalho, esta técnica é investigada apresentando operadores aritméticos que usam o processo de codificação como um método de redução da atividade de chaveamento. Para esta tarefa, propomos um código Híbrido que produz operadores aritméticos eficientes na redução do consumo de potência. Também são apresentadas as condições nas quais estes operadores aritméticos podem ser utilizados no sentido de reduzir a atividade de chaveamento em barramentos de dados.

Outro objetivo do presente trabalho é a investigação de técnicas que podem ser utilizadas para a redução do consumo de potência em arquiteturas de filtro FIR e FFT dedicadas. Dada a natureza dos algoritmos de filtro FIR e FFT, que apresentam a multiplicação de dados por apropriados coeficientes, uma das técnicas mais utilizadas para redução de potência nestes algoritmos consiste na manipulação dos coeficientes no sentido de reduzir a atividade de chaveamento na entrada dos circuitos multiplicadores. Neste trabalho, este aspecto é abordado a partir do estudo do melhor posicionamento dos coeficientes para redução do consumo de potência nestes circuitos. Neste contexto, propomos um algoritmo para ordenação e particionamento dos coeficientes utilizados nas arquiteturas propostas. Este algoritmo representa uma extensão da técnica de ordenação de coeficientes proposta em [MEH 96]. No algoritmo proposto em [MEH 96], a ordenação dos coeficientes só permite a aplicação da técnica em circuitos de filtros FIR seqüenciais. No nosso algoritmo, o aspecto do particionamento, além da ordenação, permite a aplicação da técnica em arquiteturas dedicadas semi-paralelas. Neste capítulo será apresentado o impacto, em redução de potência, da aplicação das técnicas de manipulação de coeficientes em arquiteturas seqüenciais e semi-paralelas dos algoritmos de filtro FIR de 8 *taps* e FFT de fator comum na base 2 de 16 pontos. Estas arquiteturas serão apresentadas em detalhe no Capítulo 7.

5.1 Codificação para Baixa Potência

Como visto na Seção 4.1.2, um dos métodos que tem sido largamente utilizado na área de redução de potência é a codificação de dados para a redução da atividade de chaveamento em barramentos. De fato, em diversas aplicações a atividade de chaveamento em barramentos com valor elevado de carga capacitiva contribui para uma grande parte do consumo de potência total. Recentes pesquisas têm dado especial atenção à codificação dos sinais nos barramentos de endereços

no sentido de reduzir a atividade de transição [STA 95, STA 97a, STA 97, BEN 98, MUR 98, HAK 99, RAM 99, CHA 2000, CHE 2000, HEN 2001, AGH 2002], onde são reportadas reduções na atividade de transição nos barramentos com reduções de potência entre 33% e 75% com o uso de diferentes métodos de codificação. Entretanto, nenhum destes trabalhos aborda o aspecto da utilização dos códigos para a implementação de operadores aritméticos. No presente trabalho, propomos a implementação de operadores aritméticos que possam operar diretamente com diferentes códigos aplicados às suas entradas evitando o *hardware* adicional para codificação/decodificação dos dados.

Apesar dos barramentos de dados em geral não apresentarem o mesmo grau de seqüencialidade dos barramentos de endereços, existem muitos casos onde a correlação dos sinais de dados é significativa e pode ser explorada no sentido da sua redução de potência. Além do aspecto relacionado com a otimização da atividade de transição nos barramentos, a codificação dos dados pode reduzir a própria complexidade dos módulos aritméticos, de acordo com a estrutura dos operandos.

No processo de implementação de operadores aritméticos que operam diretamente com entradas codificadas, investigam-se algumas técnicas apresentadas no Capítulo 4.

5.1.1 Técnica *Transition Coding*

A técnica *Transition Coding* é utilizada para transformar a representação do código do barramento antes que a operação aritmética seja realizada. As informações a serem codificadas ou decodificadas são comparadas com os dados anteriores que são armazenados.

A idéia de implementação de operadores aritméticos operando diretamente com os dados na representação *Transition Coding* deve considerar a seqüência de operações efetuadas em cada instante de tempo. A Tabela 5.1 mostra um exemplo de uma seqüência de operação de soma de 1 bit utilizando a representação *Transition Coding*.

Na Tabela 5.1 estão representadas todas as possibilidades de transições das entradas A e B. As variáveis com índice (T) são responsáveis pela indicação da ocorrência de transições entre os estados anterior e atual.

Apresentando a linha 3 da Tabela 5.1 como exemplo, observa-se as entradas A e B com valores lógicos iniciais iguais a 0 e 1 respectivamente. Neste caso, a saída de soma S apresenta um valor igual a 1 e a saída de *carry* C_{out} apresenta valor lógico igual a 0. Para novos valores de dados, observa-se a entrada A com mesmo valor lógico anterior e a entrada B passando a receber um novo valor lógico igual a 0. Desta forma, a entrada A permanece inalterada, portanto $A_{(T)} = 0$, enquanto que a entrada B passa a apresentar novo valor lógico, portanto $B_{(T)}$=1, apontando que houve uma transição na sua entrada. Com a nova soma, a saída S passa do valor lógico 1 anterior para o valor lógico 0. Assim sendo, a saída de soma $S_{(T)}$ aponta um valor de transição assumindo valor lógico 1. Entretanto, para esta operação de soma, a saída de *carry* permanece com seu valor inalterado e a variável $C_{out(T)}$ não aponta nenhuma transição permanecendo com valor lógico 0.

Na operação de soma em codificação *Transition*, os dados de saída dependem dos estados anterior e atual e esta informação na saída não é suficiente. Verificando como exemplo as linhas 10 e 11 da Tabela 5.1, observa-se que embora as entradas $A_{(T)}$ e $B_{(T)}$ apresentem os mesmos valores lógicos nestas duas linhas, a saída de

TABELA 5.1 – Somador completo na representação *Transition Coding*.

Linhas	A	B	$A_{(T)}$	$B_{(T)}$	S	$S_{(T)}$	C_{out}	$C_{out(T)}$
1	0→0	0→0	0	0	0→0	0	0→0	0
2	0→0	0→1	0	1	0→1	1	0→0	0
3	0→0	1→0	0	1	1→0	1	0→0	0
4	0→0	1→1	0	0	1→1	0	0→0	0
5	0→1	0→0	1	0	0→1	1	0→0	0
6	0→1	0→1	1	1	0→0	0	0→1	1
7	0→1	1→0	1	1	1→1	0	0→0	0
8	0→1	1→1	1	0	1→0	1	0→1	1
9	1→0	0→0	1	0	1→0	1	0→0	0
10	1→0	0→1	1	1	1→1	0	0→0	0
11	1→0	1→0	1	1	0→0	0	1→0	1
12	1→0	1→1	1	0	0→1	1	1→0	1
13	1→1	0→0	0	0	1→1	0	0→0	0
14	1→1	0→1	0	1	1→0	1	0→1	1
15	1→1	1→0	0	1	0→1	1	1→0	1
16	1→1	1→1	0	0	0→0	0	1→1	0

carry $C_{out(T)}$ apresenta valores lógicos diferentes. Isto ocorre pelo fato desta saída depender dos seus estados atual e anterior. Levando este aspecto em consideração, observa-se que os valores para cada operando requerem um *hardware* complexo para realizar as operações aritméticas, o que torna inviável a sua implementação como operador aritmético de baixa potência.

5.1.2 Código Redundante

Na tentativa de reduzir a complexidade dos operadores aritméticos e conseqüentemente a sua atividade de transição, foi investigada a viabilidade de utilização de um código redundante. Este código utiliza um bit extra com o objetivo de tentar minimizar a complexidade lógica dos módulos e a atividade de transição, como mostra o exemplo da Tabela 5.2.

TABELA 5.2 – Exemplo de código Redundante.

Código	Decimal
000	0
001	1
011	X
010	2
110	X
111	X
101	X
100	3

O exemplo da Tabela 5.2 mostra que o código Redundante utiliza 3 bits para

TABELA 5.3 – Área e potência para somadores em códigos Binário e Redundante.

Somadores (0 a 3)	Literais	Potência (μW)	
		Uniforme	Vetores
Binário	28	84	72
Redundante	93	295,7	240,2
Dif(Redundante→Binário)	+232%	+252%	+233%

a representação dos valores decimais entre 0 e 3. Os códigos cuja representação decimal são assinaladas por um X indicam que estes podem assumir qualquer valor lógico. Logo, os códigos relativos a estes valores indefinidos são associados aos demais códigos representativos para o processo de minimização da complexidade do circuito lógico resultante.

Como pode ser observado na Tabela 5.2, o número de possibilidades de combinações lógicas para este tipo de codificação aumenta consideravelmente. Para um código Redundante de valores até 2^n codificados com m bits o número de combinações de códigos é dado por $\binom{2^m}{2^n}$. Para o caso particular mostrado na Tabela 5.2, onde m=3, pode-se ter um total de $\binom{8}{4} = 1680$ possibilidades de combinações de códigos diferentes. A Tabela 5.3 mostra os resultados de área e potência obtidos no ambiente SIS, para operadores aritméticos de soma com números decimais de 0 a 3 em códigos Binário e Redundante do exemplo da Tabela 5.2. Na Tabela 5.3, o número de literais representa uma estimativa da quantidade de transistores utilizada pelos circuitos, onde para cada literal estão associados dois transistores. O valor de potência é obtido com a ferramenta *power estimate* do ambiente SIS para dois tipos de sinais. Um destes sinais representa uma distribuição de entrada uniforme, enquanto que o outro representa uma seqüência de valores de todas as combinações entre 0 e $2^n - 1$.

Na Tabela 5.3 e demais, onde a potência é estimada pela ferramenta *power estimate*, utiliza-se uma freqüência padrão de 20 MHz para o relógio e para os sinais de atividade máxima e considera-se um modelo de atraso zero, a menos que se indique em contrário.

Como pode ser observado pelos resultados da Tabela 5.3, o operador aritmético de soma utilizando o exemplo de codificação da Tabela 5.2 mostra uma grande complexidade em termos de área comparado ao operador em código Binário. Isto devido ao fato de ser utilizado um maior número de entradas para o operador aritmético que opera diretamente com este esquema de codificação. Esta maior complexidade se reflete em um maior consumo de potência para o operador em código Redundante, como mostrado na Tabela 5.3.

Algumas outras combinações de códigos Redundantes foram investigadas. No entanto, as suas implementações não mostraram eficiência em redução de potência comparado aos operadores Binários.

5.1.3 Código Gray

Como discutido no Capítulo 4 (Seção 4.1.2.3), o código Gray é atrativo para baixa potência devido a sua característica intrínseca de baixa atividade de chaveamento. Um problema inicial identificado para a utilização do código Gray em op-

eradores aritméticos se refere ao fato de que todos os bits na saída da operação aritmética são função de todos os bits de todas as entradas. Isto ocorre de maneira contrária ao código Binário, onde seus bits menos significativos dependem somente dos bits menos significativos dos operandos. Neste caso, a lógica combinacional requerida pelos módulos aritméticos usando operandos codificados em código Gray pode ser muito maior em relação aos módulos Binários, como mostra a Tabela 5.4.

TABELA 5.4 – Valores de área para somadores e multiplicadores em códigos Binário e Gray.

Códigos	Literais			
	Somador		Multiplicador	
	2 bits	4 bits	2 bits	4 bits
Binário	28	56	36	168
Gray	63	348	25	594
Dif(Gray→Binário)	+125%	+521%	-30%	+253%

Apesar da maior complexidade mostrada pelos circuitos somadores em código Gray, observa-se pelos resultados da Tabela 5.4 que é possível reduzir a complexidade lógica do multiplicador de 2 bits operando diretamente neste código. Entretanto, esta complexidade aumenta para circuitos de 4 bits, onde cada uma das saídas é função de um maior número de bits. A Tabela 5.5 mostra os resultados de consumo de potência para os operadores de 2 bits mostrados na Tabela 5.4 utilizando entradas uniformemente distribuídas e vetores de entrada com a seqüência de todas as combinações possíveis entre 0 e 2^n-1.

TABELA 5.5 – Consumo de potência para somadores e multiplicadores de 2 bits em códigos Binário e Gray.

Código	Potência(μW)			
	entrada uniforme		vetores de entrada	
	Somador	Multiplicador	Somador	Multiplicador
Bin	84	64	72	56
Gray	221	73	124	44
Dif(Gray→Binário)	+163,1%	+13,8%	+72,2%	-21,4%

Como pode ser observado na Tabela 5.5, a maior complexidade mostrada pelos circuitos somadores em código Gray de 2 bits faz com que estes operadores apresentem maiores consumos de potência. Para a operação de multiplicação de 2 bits em código Gray, que apresenta uma menor complexidade lógica (como mostrado na Tabela 5.4), observa-se uma redução de potência para vetores de entrada com a seqüência de todas as combinações possíveis. Entretanto, para circuitos de 4 bits, onde a complexidade de *hardware* requerida pelos operadores em código Gray se torna maior (como mostrado na Tabela 5.4), tem-se maiores valores de consumo de potência envolvidos, como mostra a Tabela 5.6.

Pelos resultados mostrados pela Tabela 5.6, observa-se que o elevado consumo de potência apresentado pelos operadores aritméticos de 4 bits em código Gray inviabiliza a sua utilização para projetos de baixa potência. Esta situação é pior para

TABELA 5.6 – Consumo de potência para somadores e multiplicadores de 4 bits em códigos Binário e Gray.

Código	Potência(μW)			
	entrada uniforme		vetores de entrada	
	Somador	Multiplicador	Somador	Multiplicador
Bin	167,5	368,7	109,5	196,9
Gray	1158	1823	332	678
Dif(Gray→Binário)	+591%	+394%	+203%	+244%

um maior número de bits. Outro aspecto limitante à utilização do código Gray em operadores aritméticos diz respeito à impossibilidade de desenvolver estruturas regulares de operadores aritméticos de tamanhos diferentes (4 bits, 8 bits, 16 bits, 32 bits, etc.). Entretanto, neste trabalho é apresentado um código que é um compromisso entre os códigos Binário e Gray, denominado código Híbrido, como será mostrado a seguir. Um código similar é utilizado em [HAK 99], mas a abordagem se limita à segmentação do código Gray em 4 bits para aplicação em barramentos de endereços de microcontroladores.

5.1.4 Código Híbrido

A idéia do código Híbrido é dividir os operandos em grupos de m bits, codificar cada grupo utilizando o código Gray e utilizar o comportamento do código Binário para propagar o *carry* entre os grupos. Desta forma, o número de transições em cada grupo pode ser minimizado e uma estrutura regular pode ser obtida, onde os grupos menos significativos do resultado dependem somente dos grupos menos significativos dos operadores. A Tabela 5.7 mostra as representações dos códigos Binário, Híbrido (m=2) e Gray para números de 4 bits.

Como pode ser observado pela Tabela 5.7, o código Híbrido representa um compromisso entre a mínima dependência das entradas de dados apresentada pelo código Binário e a característica de baixa atividade de chaveamento apresentada pelo código Gray. A Tabela 5.8 apresenta o número de transições dos códigos Binário, Gray e Híbrido para uma determinada seqüência de contagem.

Como pode ser observado pela Tabela 5.8, o código Híbrido é uma situação intermediária entre os códigos Binário e Gray em termos de número de transições. Desta forma, para sistemas onde a capacitância chaveada no barramento de dados é significativa e onde os dados apresentam um alto grau de correlação, a utilização do código Híbrido pode reduzir o consumo de potência em até cerca de um terço em relação ao código Binário. O número de transições para seqüências de contagem, como apresentadas na Tabela 5.8, é calculado de acordo com as Equações 5.1, 5.2 e 5.3 para os códigos Binário, Gray e Híbrido respectivamente.

$$\text{n}^\text{o}\text{\_de\_trans.(Bin)} = 2^{n+1} - 2 \tag{5.1}$$

$$\text{n}^\text{o}\text{\_de\_trans.(Gray)} = 2^{n} \tag{5.2}$$

$$\text{n}^\text{o}\text{\_de\_trans.(Hib)} = 2^{m}\frac{2^{n}-1}{2^{m}-1} \tag{5.3}$$

TABELA 5.7 – Representações dos códigos Binário, Híbrido (m=2) e Gray.

Decimal	Binário	Híbrido (m=2)	Gray
0	0000	0000	0000
1	0001	0001	0001
2	0010	0011	0011
3	0011	0010	0010
4	0100	0100	0110
5	0101	0101	0111
6	0110	0111	0101
7	0111	0110	0100
8	1000	1100	1100
9	1001	1101	1101
10	1010	1111	1111
11	1011	1110	1110
12	1100	1000	1010
13	1101	1001	1011
14	1110	1011	1001
15	1111	1010	1000

TABELA 5.8 – Número de transições para os códigos Binário, Híbrido (m=2) e Gray.

Número de Bits	Número de Transições			Diferença	
	Binário	Gray	Híbrido ($m = 2$)	Hib→Bin	Hib→Gray
$n = 4$ bits (0,1,..15,0)	30	16	20	-33,3%	+25%
$n = 8$ bits (0,1,..255,0)	510	256	340	-33,3%	+32,8%
$n = 16$ bits (0,1,..65535,0)	131070	65536	87380	-33,3%	+33,3%

A Equação 5.3 apresenta a forma genérica do cálculo do número de transições em código Híbrido. Para o caso particular, m=2, este cálculo é efetuado de acordo com a Equação 5.4.

$$\text{n}^{\text{o}}\text{\_de\_trans.(Hib, m} = 2) = \frac{4}{3}(2^n - 1) \tag{5.4}$$

Além da redução do número de transições, uma outra vantagem apresentada pelo código Híbrido é a possibilidade de geração de circuitos somadores e multiplicadores com estruturas regulares, como será mostrado na próxima seção.

Outra característica apresentada pelo código Híbrido com $m = 2$ é a facilidade de mudança de representação para o código Binário, como mostra o exemplo da Figura 5.1. Desta forma, o processo de codificação/decodificação dos dados utiliza um *hardware* de reduzida complexidade com uma porta EXOR ligada a cada grupo de $m = 2$ bits. Neste caso, o código Híbrido também pode ser utilizado como método

TABELA 5.9 – Área e potência para somadores Binário e Híbrido de 2 bits.

Somadores 2 bits	Literais	Potência (μW)	
		Uniforme	Vetores
Binário	28	84	72
Híbrido	57	170	131
Diff(Híbrido→Binário)	+103,6%	+102,4%	+81,9%

de codificação para os barramentos de endereços.

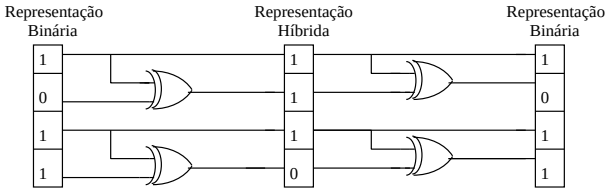

FIGURA 5.1 – Conversão entre os códigos Binário e Híbrido.

5.2 Operadores Aritméticos em Código Híbrido sem Sinal

Nesta parte do trabalho são apresentadas as estruturas dos operadores básicos que são utilizados para a construção de circuitos multiplicadores Híbridos *array*, assumindo uma representação de operandos sem sinal. Nesta seção será abordada a utilização de código Híbrido que utiliza grupos de bits de tamanho $m = 2$. Posteriormente, será mostrada a geração de circuitos com tamanhos de m maiores do que 2.

5.2.1 Elementos Aritméticos Básicos

Os elementos básicos apresentados nesta seção foram desenvolvidos no formato BLIF, mapeado para a biblioteca `mcnc`, conforme Seção 2.4.3. Os valores de potência dos elementos básicos são obtidos a partir da utilização de sinais com distribuição de entrada uniforme e seqüência de valores de todas as combinações entre 0 e $2^n - 1$, onde n representa o número de bits da palavra.

5.2.1.1 Elemento Básico Somador Híbrido com m=2

A implementação de circuitos somadores operando diretamente em código Híbrido possibilita a obtenção de estruturas regulares e a síntese de estruturas de somadores maiores do que as estruturas mostradas em código Gray. Entretanto, observa-se que os circuitos somadores operando diretamente em código Híbrido apresentam uma maior complexidade comparados aos circuitos Binários com maiores valores de área e consumo de potência, como mostra a Tabela 5.9.

A Tabela 5.9 mostra que embora o efeito da aplicação da seqüência de vetores de entrada reduza a diferença de consumo de potência entre os circuitos Híbrido e

Binário, este valor ainda é bastante significativo devido à complexidade do somador Híbrido.

Como pôde ser observado na Figura 5.1, a conversão entre os códigos Binário e Híbrido é realizada com reduzida complexidade de *hardware* adicional. Neste caso, a forma mais eficiente de implementação de um somador Híbrido é através do uso de um somador Binário após a conversão das entradas, como mostra a Figura 5.2. Naturalmente, ainda assim isto implica uma maior área e um maior consumo de potência dos somadores Híbridos em relação aos somadores Binários, como mostra a Tabela 5.10. Este fato ocorre devido às portas lógicas EXOR necessárias no circuito somador Híbrido. Entretanto, dependendo da correlação dos dados e dos valores de capacitância de carga, é possível uma redução de potência global mesmo considerando estes circuitos, como será mostrado mais adiante na aplicação deste operador Híbrido em circuitos multiplicadores tipo *array*.

TABELA 5.10 – Comparação de área e potência entre somador Binário e o novo somador Híbrido de 2 bits.

Somadores 2 bits	Literais	Potência (μW)	
		Uniforme	Vetores
Binário	28	84	72
Híbrido	40	121,2	103,6
Dif(Híbrido→Binário)	+42,8%	+44,3%	+44,0%

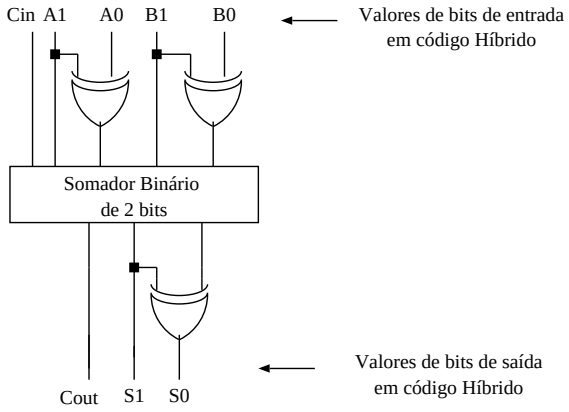

FIGURA 5.2 – Circuito somador Híbrido de 2 bits.

5.2.1.2 Elemento Básico Multiplicador Híbrido com m=2

Para o circuito multiplicador Híbrido é possível gerar uma simples estrutura de 2 bits utilizando apenas 8 portas lógicas, como mostra a Figura 5.3. Esta estrutura apresenta menores valores de área e potência em relação à estrutura Binária, como mostram os resultados da Tabela 5.11.

Ao contrário do apresentado para os circuitos somadores básicos, observa-se que o multiplicador Híbrido pode ser implementado sem que seja necessária nenhuma mudança de representação no código de entrada. Como será visto nas próximas

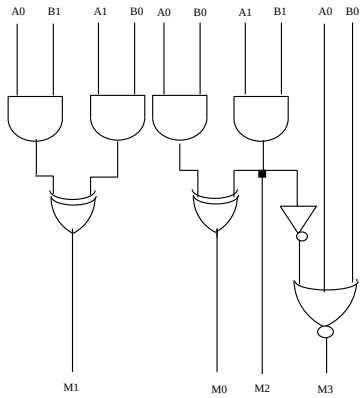

FIGURA 5.3 – Circuito multiplicador Híbrido de 2 bits.

TABELA 5.11 – Área e potência para os multiplicadores Binário e Híbrido de 2 bits.

Multiplicadores 2 bits	Literais	Potência (μW)	
		Uniforme	Vetores
Binário	36	64,1	56,0
Híbrido	20	55,0	36,7
Dif(Híbrido→Binário)	-44,4%	-14,2%	-34,5%

seções, o multiplicador Híbrido básico mostrado na Figura 5.3 é utilizado nos multiplicadores tipo *array* para produzir os termos dos produtos parciais.

5.2.2 Multiplicador Híbrido tipo *Array*

A operação de multiplicação paralela é realizada a partir da soma de seqüências de produtos parciais. Os produtos parciais são gerados por células de multiplicação intermediárias e o *carry* é propagado para cada grupo [HAR 95]. Neste trabalho, são implementadas arquiteturas de circuitos multiplicadores *array* nas formas Binária e Híbrida. As arquiteturas Binárias são implementadas utilizando estruturas de multiplicador tipo *array* conhecidas da literatura [HWA 79, HAR 95] e as arquiteturas Híbridas efetuam operações de multiplicações parciais a cada 2 bits, em uma representação na base 4. Essa representação permite a implementação de estruturas regulares, como será mostrado posteriormente. Mais adiante serão apresentados resultados para maiores valores de m.

5.2.2.1 Multiplicador *Array* Binário

Na operação de um multiplicador tipo *array* Binário, os produtos parciais são realizados em uma forma paralela. A multiplicação de números binários positivos é realizada da mesma forma em que a operação é realizada com números decimais.

Considere dois operandos de largura W-bits, sendo dados por: $A = \sum_{i=0}^{W-1} a_i 2^i$ and $B = \sum_{j=0}^{W-1} b_j 2^j$. Tem-se que:

$$A \times B = \sum_{j=0}^{W-1} A \cdot b_j 2^j \tag{5.5}$$

onde o termo $A \cdot b_j$ é dado por:

$$A \cdot b_j = \sum_{i=0}^{W-1} b_j \cdot a_i 2^i \tag{5.6}$$

Um multiplicador tipo *array* traduz estas expressões diretamente para *hardware*, como pode ser observado na Figura 5.4 para W=4 bits. Tem-se as W linhas de produtos parciais da Equação 5.5, cada uma composta de produtos de níveis de W bits, como na Equação 5.6, que pode ser arranjada em uma simples e regular estrutura do tipo *array*. Cada produto de bit é simplesmente uma porta lógica AND.

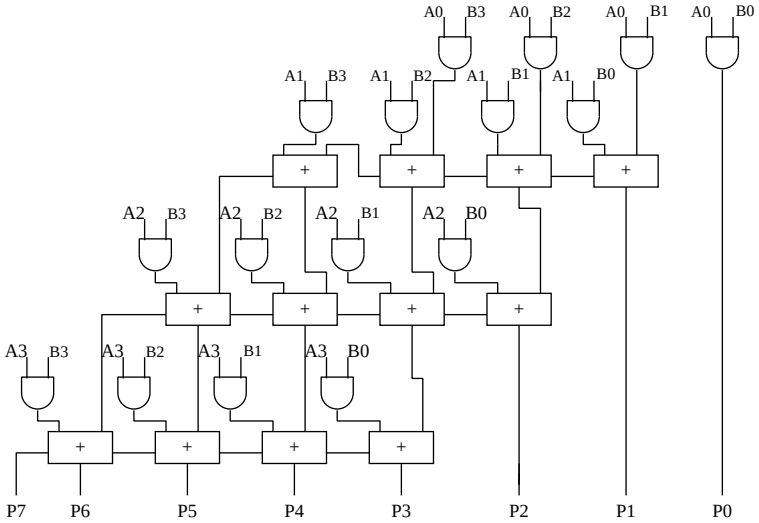

FIGURA 5.4 – Multiplicador *array* Binário de 4 bits.

Como observado na Figura 5.4, no multiplicador *array* a operação é realizada pela soma de produtos parciais em seqüência. Os produtos parciais são obtidos com portas lógicas AND a partir das operações $1 \times W$. Os W bits menos significativos do produto são produzidos no lado direito da estrutura *array*.

O algoritmo apresentado na Figura 5.5 foi implementado no sentido de gerar multiplicadores tipo *array* Binário no formato BLIF para qualquer tamanho de palavra W. Como visto na Figura 5.5, os multiplicadores *array* Binários são obtidos a partir da especificação em formato BLIF, dos termos dos produtos parciais (portas AND) e operadores somadores. A quantidade de operadores necessários em cada arquitetura é calculada de acordo com o número de bits na palavra, representado pela variável W, que define o número de colunas e linhas dos elementos necessários para composição do multiplicador completo. O resultado gerado pelo algoritmo é um circuito em formato BLIF que permite estimar valores de área, atraso e potência no ambiente SIS para cada comprimento de palavra. Os valores destes parâmetros são utilizados para comparação com a correspondente arquitetura Híbrida.

***Binary Multiplier Procedure* (*W*)**

```
1.  /* First line */
2.  for j=0 to W-1 {
3.     write AND(in(A[0],B[j]); out(P[0,j]));
4.  }
5.  /* Remaining lines */
6.  for i=1 to W-1{
7.     for j=0 to W-1 {
8.        write AND(in(A[i],B[j]); out(R[i,j]));
9.        write adder(in(P[i-1,j+1],R[i,j],c_in[i,j-1]); out(P[i,j],c_out[i,j]));
10.    }
11.    P[i,j+1]=c_out[i,j];
12. }
```

FIGURA 5.5 – Algoritmo para a geração de diferentes multiplicadores Binários.

5.2.2.2 Operação do Multiplicador Híbrido com m=2

No multiplicador Híbrido tipo *array*, a operação é realizada em cada grupo de m bits, onde m representa o tamanho do grupo para o código Híbrido. Nesta seção, o enfoque é o caso onde m=2. Os termos dos produtos parciais são expressos na representação base 4, como mostrado na Figura 5.6, para um exemplo de uma operação de multiplicação de 4 bits do tipo *array* em código Híbrido.

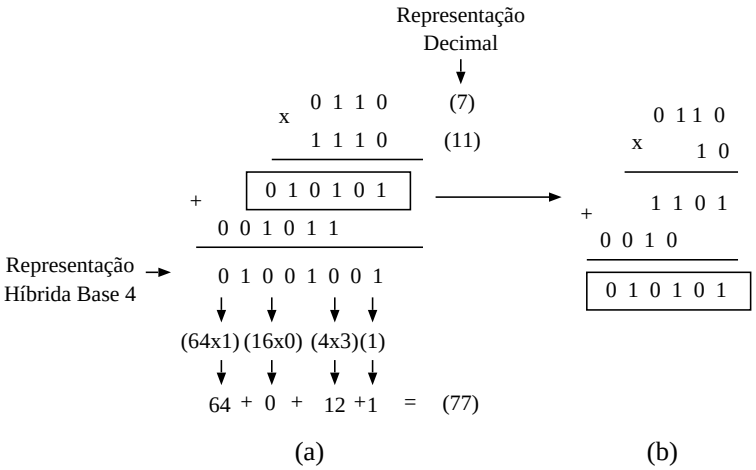

FIGURA 5.6 – Operação de multiplicação de 4 bits em código Híbrido.

Como mostrado na Figura 5.6, os termos dos produtos parciais são obtidos pela multiplicação de cada grupo de 2 bits dos termos multiplicador e multiplicando. Cada linha parcial é obtida por operações $2 \times W$, como mostrado na Figura 5.6(b), onde W representa o tamanho da palavra do termo multiplicando.

A linha final para a operação de multiplicação em código Híbrido é obtida a partir da soma de cada grupo de 2 bits dos termos dos produtos parciais, como mostrado na Figura 5.6(a). Os valores dos resultados em decimal são obtidos pela conversão de cada grupo de 2 bits, assumindo uma representação em base 4.

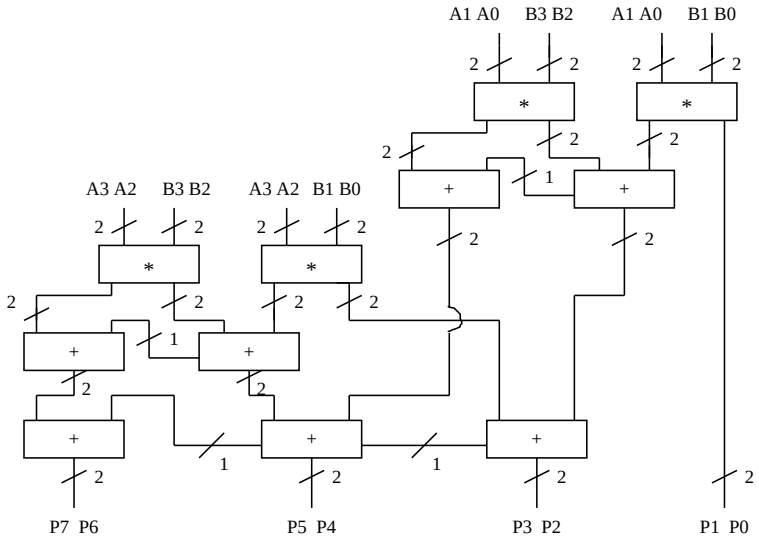

FIGURA 5.7 – Arquitetura do multiplicador Híbrido *array* de 4 bits.

5.2.2.3 Arquitetura do Multiplicador Híbrido com m=2

Da mesma forma como discutido para os multiplicadores Binários do tipo *array*, a operação de multiplicação em código Híbrido pode ser realizada de uma forma regular, como mostrado na Figura 5.6. Entretanto, ao contrário do multiplicador Binário, onde as operações são efetuadas bit a bit, as operações Híbridas são realizadas em grupos de m bits de cada vez. Para a implementação de uma arquitetura que possa considerar estes aspectos, são utilizados os elementos básicos somador e multiplicador mostrados nas Figuras 5.2 e 5.3. A arquitetura de um multiplicador Híbrido tipo *array* é mostrada na Figura 5.7 para operação de 4 bits e m=2.

Como pode ser visto pela Figura 5.7, o multiplicador Híbrido *array* apresenta uma estrutura regular, com os elementos básicos sendo utilizados para realizar os produtos parciais. Os dois bits menos significativos são produzidos imediatamente após a primeira multiplicação $2 \times W$, mostrada na Figura 5.6(b). Os demais bits são obtidos pela adição dos termos dos produtos parciais. Como pode ser observado na Figura 5.7, utiliza-se apenas uma linha de circuitos somadores responsáveis pelas somas dos termos dos produtos parciais. Para uma arquitetura em código Híbrido de W bits, serão utilizadas (W/2)-1 destas linhas de somadores, como será mostrado na próxima seção.

Outro aspecto a ser destacado na arquitetura do multiplicador Híbrido, mostrada na Figura 5.7, é que alguns circuitos somadores são mais simples, uns por não necessitarem de *carry* de entrada e outros por não necessitarem de *carry* de saída. Nos somadores dos módulos dos produtos parciais, a operação de soma é efetuada com a utilização dos 2 bits mais significativos dos multiplicadores que efetuam as operações $A1A0 \times B3B2$ e $A3A2 \times B3B2$ e com o *carry* gerado pelo circuito somador imediatamente anterior. Dado que estes circuitos multiplicadores podem gerar no máximo valores lógicos iguais a 11 nos seus bits mais significativos ($3 \times 3 = 9 = 1101$), logo a adição deste valor com o *carry* gerado pelo somador imediatamente anterior gera no máximo um valor igual a 10 (3 em decimal) nas saídas de cada somador de cada módulo, garantindo assim c_{out}=0.

Para o somador que faz parte da linha responsável pela soma dos produtos

parciais, a operação de soma é efetuada utilizando o valor de *carry* gerado pelo somador imediatamente anterior desta linha com os valores dos 2 bits do circuito somador mais simples que faz parte do segundo módulo dos produtos parciais. No caso em que a operação de multiplicação do circuito envolve os maiores valores decimais possíveis para esta arquitetura ($15 \times 15 = 225$), a operação de soma envolvendo os valores gerados por estes circuitos somadores proporciona um resultado nas saídas *P7P6* do circuito somador igual a 10 ($1010 \times 1010 = 10110101$). Desta forma, observa-se que o circuito somador que produz os bits mais significativos do resultado de multiplicação nunca gera um valor de *carry* de saída.

5.2.2.4 Algoritmo para Geração dos Multiplicadores Híbridos com m=2

A partir da arquitetura de multiplicador Híbrido de 4 bits mostrada na Figura 5.7, pode-se considerar a geração de arquiteturas de diferentes tamanhos a partir da sua estrutura regular. A Figura 5.8 apresenta um pseudo-código que permite a geração de arquiteturas de multiplicadores Híbridos para qualquer tamanho de palavra.

Hybrid Multiplier Procedure **(W)**

1. for i=0 to W-1, i=i+2{
2. for j=0 to W-1, j=j+2{
3. write multiplier(in($A[i+1]A[i]$,$B[j+1]B[j]$);
   out($P[i/2,2j+3]P[i/2,2j+2]P[i/2,2j+1]P[i/2,2j]$));
4. }
5. for j=0 to W-1, j=j+2{
6. write adder(in($P[i/2,2j+5]P[i/2,2j+4]$,$P[i/2,2j+3]P[i/2,2j+2]$,$c_{in}[j/2]$);
   out($R[i/2,j+3]R[i/2,j+2]$,$c_{out}[(j/2)+1]$));
7. }
8. }
9. for i=2 to W-1, i=i+2{
10. $R[i/2,1]R[i/2,0]$=$P[i/2,1]P[i/2,0]$;
11. for j=0 to W, j=j+2{
12. write adder(in($R[(i/2)\text{-}1,j+3]R[(i/2)\text{-}1,j+2]$,$R[i/2,j+1]R[i/2,j]$,$c_{in}[j/2]$);
    out($P[i/2,j+1+i]P[i/2,j+$i$]$,$c_{out}[(j/2)+1]$));
13. $R[i/2,j+1]R[i/2,j]$=$P[i/2,j+1+i]P[i/2,j+i]$;
14. }
15. }

FIGURA 5.8 – Algoritmo para geração de multiplicadores Híbridos de diferentes comprimentos de palavra.

O pseudo-código apresentado na Figura 5.8 mostra que as estruturas Híbridas são obtidas pela representação em formato BLIF dos termos dos produtos parciais e somadores. O número de colunas e linhas compostas pelos elementos aritméticos básicos que define a arquitetura completa são calculados de acordo com o número de bits representado pela variável W.

Apesar do algoritmo da Figura 5.8 mostrar arquiteturas de multiplicadores em código Híbrido, que realizam operações em cada grupo de 2 bits, observa-se que se pode adaptar a sua representação para a realização de operações em cada grupo de m bits. Neste caso, torna-se necessário a utilização de elementos básicos

TABELA 5.12 – Área, em literais, para os multiplicadores *array* Binário e Híbrido.

Número de Bits	Binário	Híbrido	Dif Hib→Bin
4	168	232	+38,1%
8	848	1144	+34,9%
12	2040	2728	+33,7%
16	3744	4984	+33,1%
32	15680	20728	+32,1%

somadores e multiplicadores na representação da base 2^m. Um algoritmo semelhante foi desenvolvido para valor de m genérico. As estruturas resultantes deste algoritmo serão mostradas posteriormente.

5.2.3 Comparação de Desempenho dos Circuitos Multiplicadores *Array*

Como mostrado na seção anterior, os algoritmos das Figuras 5.5 e 5.8 permitem a implementação de circuitos multiplicadores Binários e Híbridos para qualquer tamanho de palavra. Nesta seção são mostrados os resultados de área, atraso e potência para arquiteturas de multiplicadores Binários e Híbridos de 4, 8, 12, 16 e 32 bits. Os resultados de área e atraso são obtidos no ambiente SIS. A área é estimada a partir do número de literais dos circuitos. Os resultados de atraso são obtidos a partir da soma dos atrasos caracterísiticos (atraso genérico da biblioteca `mcnc`) de cada porta individual presente no caminho crítico dos circuitos. Os resultados de consumo de potência são apresentados com valores obtidos com a ferramenta SIS, com modelo de atraso zero, e com a ferramenta SLS, que leva em consideração os efeitos de *glitchings* que são muito comuns em circuitos do tipo multiplicadores.

5.2.3.1 Resultados de Área

As arquiteturas de circuitos multiplicadores Binário e Híbrido *array* apresentam termos de produtos parciais expressos em formas diferentes. Enquanto que a arquitetura Binária utiliza $W \times W$ portas AND, as arquiteturas Híbridas utilizam $W/2 \times W/2$ elementos multiplicadores básicos, como mostrado na Figura 5.3.

Apesar das arquiteturas Híbridas utilizarem metade dos elementos de produto básicos comparado às arquiteturas Binárias, observa-se que cada elemento multiplicador Híbrido básico é composto por 8 portas lógicas, como mostrado na Figura 5.3. Desta forma, os multiplicadores Híbridos apresentam uma maior quantidade de portas lógicas em relação aos multiplicadores Binários, para realizar os produtos parciais. Além disto, como mostrado na Tabela 5.10, os elementos somadores Híbridos básicos de 2 bits apresentam maior área em relação aos elementos Binários. Desta forma, como mostrado pela Tabela 5.12, os multiplicadores Híbridos *array* apresentam maior área em relação aos multiplicadores Binários *array* para qualquer tamanho de palavra.

5.2.3.2 Resultados de Atraso

Apesar dos multiplicadores Híbridos apresentarem maior área, estas arquiteturas podem apresentar menor valor de atraso em relação aos multiplicadores Binários tipo *array*. Como mostrado na Tabela 5.13, para arquiteturas de circuitos mais complexas (de maior comprimento de palavras), os multiplicadores Híbridos apresentam menores valores de atraso em relação às arquiteturas Binárias. Isto ocorre devido ao fato dos multiplicadores Híbridos apresentarem menor caminho crítico em relação aos multiplicadores Binários. Como mostrado nas Figuras 5.5 e 5.8, enquanto que a arquitetura Híbrida apresenta ($W/2$)-1 linhas de somadores responsáveis pela adição dos termos dos produtos parciais, a arquitetura Binária apresenta W-1 destas linhas de somadores. Este fator se torna expressivo em termos de valores de atraso apresentados por arquiteturas com maiores tamanhos de palavras.

TABELA 5.13 – Atrasos para multiplicadores *array* Binário e Híbrido.

Número de Bits	Atraso (ns)		Diferença Hib→Bin
	Binário	Híbrido	
4	46,4	47,1	+1,5%
8	117,6	104,7	-10,9%
12	188,8	162,3	-14,0%
16	260,0	219,9	-15,4%
32	544,8	450,3	-17,3%

A Tabela 5.13 mostra que para um multiplicador de 4 bits, o circuito Binário apresenta menor valor de atraso em relação ao código Híbrido. Entretanto, em circuitos mais complexos de 8 a 32 bits, onde o caminho crítico se torna mais expressivo, os circuitos Híbridos apresentam menores valores de atraso.

5.2.3.3 Resultados de Consumo de Potência

Apesar da maior complexidade apresentada pelos multiplicadores Híbridos tipo *array* com maiores valores de área, como mostrado na Tabela 5.12, estes circuitos podem apresentar um menor consumo de potência em relação às arquiteturas Binárias, como mostrado na Tabela 5.14. Isto ocorre devido ao fato dos multiplicadores Híbridos apresentarem um menor número de transições internas em relação aos multiplicadores Binários.

Como mostrado em [LAN 95, RAM 97, COS 99], para uma determinada seqüência de dados para um sinal conhecido nas entradas dos operadores, os bits mais significativos apresentam uma alta correlação temporal e um reduzido número de transições. Desta forma, um sinal de entrada com alta correlação aplicado aos multiplicadores Híbridos de tamanhos de palavras maiores pode reduzir o consumo de potência destas arquiteturas, como mostrado na Tabela 5.14. Na Tabela 5.14, os valores de potência são obtidos a partir da aplicação de um sinal de *trace* real às entradas das arquiteturas Binária e Híbrida. Este sinal representa o número de pontos de 2 sinais senoidais com diferença de fase de 90 graus.

A Tabela 5.14 mostra que para uma determinada arquitetura, quanto maior o número de pontos aplicado à entrada, menor o consumo de potência dos multiplicadores Binário e Híbrido. Este fato ocorre porque para um sinal de *trace* real com

TABELA 5.14 – Consumo de potência dos multiplicadores *array* Binário e Híbrido com modelo de atraso zero.

Número de Bits	Número de Pontos	Binário P (μW)	Híbrido P (μW)	Diferença Híbrido→Binário
4	100	113,4	128,8	+13,5%
	200	58,2	66,5	+14,3%
8	1000	668,7	710,5	+6,2%
	2000	350,8	371,3	+5,8%
12	20000	1085,5	1065,0	+1,8%
	40000	559.8	548,0	-2.1%
16	150000	3216,0	3040,7	-5,4%
	300000	1903,0	1749,4	-8,0%
32	4294964	14061,1	10905,4	-22,4%

maior número de pontos, o sinal senoidal apresenta uma melhor representação com menor quantidade de variações abruptas. Neste caso, quanto maior a largura de bits dos operadores *array*, os circuitos Híbridos apresentam maior redução de potência em relação aos circuitos Binários. Isto se deve ao fato de que, em um sinal com menor quantidade de variações abruptas, o código Híbrido apresenta menor número de transições em relação ao código Binário, como mostrado na Tabela 5.8.

Outro aspecto importante apresentado na Tabela 5.14 se refere ao consumo de potência dos circuitos, de acordo com o tamanho da palavra de dados. Como pode ser observado, quanto maior o número de bits, menor o consumo de potência apresentado pelos multiplicadores Híbridos. Isto se deve ao fato de que para circuitos mais complexos, palavras de dados mais largas, os bits mais significativos se tornam mais importantes e quando é aplicado à entrada destes circuitos o sinal de *trace* com menos variações abruptas, as arquiteturas Híbridas levam vantagem em relação ao aspecto da correlação.

Os resultados de potência apresentados na Tabela 5.14 foram obtidos na ferramenta *power estimate* do ambiente SIS considerando modelo de atraso zero. Entretanto, os circuitos multiplicadores produzem uma grande quantidade de transições espúrias antes de alcançar o nível lógico correto. Levando-se em consideração que este efeito é responsável por boa parte do consumo de potência global do circuito, a ocorrência destas transições não pode ser desprezada na estimativa do consumo de potência dos circuitos. A Tabela 5.15 apresenta os resultados de consumo de potência dos circuitos multiplicadores Binário e Híbrido para palavras de 4, 8, 12 e 16 bits obtidos na ferramenta SLS, considerando o efeito de propagação de *glitchings* a partir da aplicação do *trace* real às entradas dos circuitos.

Mesmo com a propagação de *glitching* nos circuitos, o efeito da correlação dos sinais de entrada faz com que as arquiteturas Híbridas mais complexas consumam menos potência em relação às arquiteturas Binárias, como observado na Tabela 5.15. Também como pode novamente ser observado, quanto maior o número de pontos utilizados nos sinais de entrada, melhor a sua representação e conseqüentemente menor o consumo de potência dos multiplicadores.

A componente de consumo de potência devido ao efeito de *glitching* é forte-

TABELA 5.15 – Potência para multiplicadores *array* Binário e Híbrido com modelo de atraso genérico.

Número de Bits	Número de Pontos	Binário P (mW)	Híbrido P (mW)	Diferença Híbrido→Binário
4	100	0,65	0,82	+26,1%
	200	0,32	0,41	+27,8%
8	1000	6,51	6,53	+0,3%
	2000	3,40	3,42	+0,5%
12	20000	13,78	11,72	-14,9%
	40000	7,10	5,83	-17,8%
16	150000	47,90	32,26	-32,6%
	300000	17,38	13,96	-19,6%

mente dependente da topologia do circuito e do vetor de entrada aplicado [FAV 95]. A propagação destes sinais é uma função da profundidade lógica do circuito. A Tabela 5.16 mostra valores de consumo de potência para os multiplicadores Binário e Híbrido de 4, 8, 12 e 16 bits obtidos na ferramenta SLS, considerando o efeito de propagação de *glitchings* nos circuitos para sinais de entrada com distribuição uniforme.

TABELA 5.16 – Potência para multiplicadores *array* Binário e Híbrido com modelo de atraso genérico e entradas uniformemente distribuídas.

Número de Bits	Número de Pontos	Binário P (mW)	Híbrido P (mW)	Diferença Hib→Bin
4	100	1,83	2,44	+33,3%
8	1000	17,15	14,98	-12,6%
12	20000	73,71	49,12	-33,4%
16	150000	106,36	83,22	-21,2%

Como pode ser observado na Tabela 5.16, mesmo para entradas uniformemente distribuídas, quando é considerado o efeito de propagação de *glitchings*, as arquiteturas Híbridas mais complexas (maiores do que 4 bits) consomem menos potência em relação às arquiteturas Binárias. Este fato ocorre pois para estas arquiteturas mais complexas, as arquiteturas Híbridas apresentam menor profundidade lógica como mostram os resultados da Tabela 5.17. Os resultados apresentados na Tabela 5.17 foram obtidos no ambiente SIS e mostram a quantidade de níveis lógicos apresentada por cada uma das arquiteturas.

Comparando os resultados das Tabelas 5.16 e 5.17, observa-se que para circuitos multiplicadores de palavras maiores do que 4 bits, a profundidade lógica das arquiteturas Híbridas são menores em relação às arquiteturas Binárias. Desta forma, mesmo para entradas uniformemente distribuídas quando é considerado o efeito da propagação de *glitchings* os multiplicadores Híbridos apresentam menor consumo de potência. Entretanto, se este efeito não é considerado, ou seja, se é utilizado modelo de atraso zero na estimativa de potência, os multiplicadores Híbridos apre-

TABELA 5.17 – Profundidade lógica para multiplicadores *array* Binário e Híbrido.

Número de Bits	Número de Níveis		Dif. Hib→Bin
	Binário	Híbrido	
4	17	17	0,0%
8	40	36	-10,0%
12	64	56	-12,5%
16	88	76	-13,6%

sentam maior consumo de potência para entradas uniformemente distribuídas. Isto ocorre pelo fato das arquiteturas Híbridas apresentarem maiores valores de área, como mostrado na Tabela 5.12.

5.2.4 Arquiteturas com Operações Aritméticas na Base 2^m

O aspecto do alto nível de regularidade mostrado pela arquitetura em código Híbrido apresentada na seção anterior, possibilita uma fácil extensão para operações com operandos de diferentes bases. Entretanto, os resultados mostrados até o presente momento para o multiplicador Binário consideraram um multiplicador tipo *array* convencional [HWA 79] onde as operações são realizadas bit a bit.

Neste trabalho, observa-se que as mesmas características dos operadores aritméticos que operam em código Híbrido podem também ser empregadas em operadores em código Binário. Neste caso, também é possível a implementação de circuitos multiplicadores em código Binário operando em grupos de m bits. Desta forma, as considerações feitas nesta seção, para multiplicadores que operam na base 2^m, podem ser adotadas para ambas as representações.

Nesta seção são mostrados os aspectos relacionados com a operação de multiplicação na base 2^m, além das arquiteturas específicas para diferentes grupos de m bits. São apresentados resultados das aquiteturas em códigos Binário e Híbrido para grupos de bits de tamanhos m=2, 4 e 8. Esses valores são comparados ao multiplicador *array* Binário convencional (m=1). Esses valores de grupos de m bits permitem a construção de estruturas regulares, para as arquiteturas de 16 bits que serão mostradas a partir desta parte do trabalho. Entretanto, para arquiteturas com tamanhos de palavras de W bits é possível utilizar outros grupos de m bits, desde que a regularidade da arquitetura seja mantida.

Da mesma forma como apresentado na seção anterior, os resultados de área e atraso são obtidos no ambiente SIS. Os valores de potência são obtidos na ferramenta SLS considerando o efeito de *glitching*.

Na seção anterior foram utilizados diferentes números de pontos para as arquiteturas de multiplicadores com diferentes números de bits. Entretanto, deve-se observar que um grande número de pontos utilizado nos sinais de entrada pode acarretar em uma grande carga computacional, principalmente em circuitos mais complexos com um maior número de entradas (como por exemplo, os multiplicadores de 16 bits utilizados neste trabalho). Desta forma, a partir desta parte do trabalho são utilizados 10000 pontos nos sinais de *trace* real e randômico como uma boa representação dos sinais de entrada, como mostrado a seguir. Este número de pontos estabelece uma maior variação entre os bits e uma menor carga computacional.

- **Estabelecimento dos Vetores de Entrada**

Como pôde ser visto nas seções anteriores, o número de pontos utilizado nos vetores de entrada dos circuitos determina uma variação nos seus consumos de potência. De fato, quanto maior o número de pontos utilizado menor o consumo de potência dos circuitos. Isto ocorre devido à maior correlação no sinal que proporciona uma menor variação abrupta entre os bits. A Figura 5.9 mostra um gráfico do consumo de potência de multiplicadores de 16 bits, Híbrido (m=2) e Binário convencional (m=1), com uma variação no número de pontos do sinal de *trace* real aplicado às entradas.

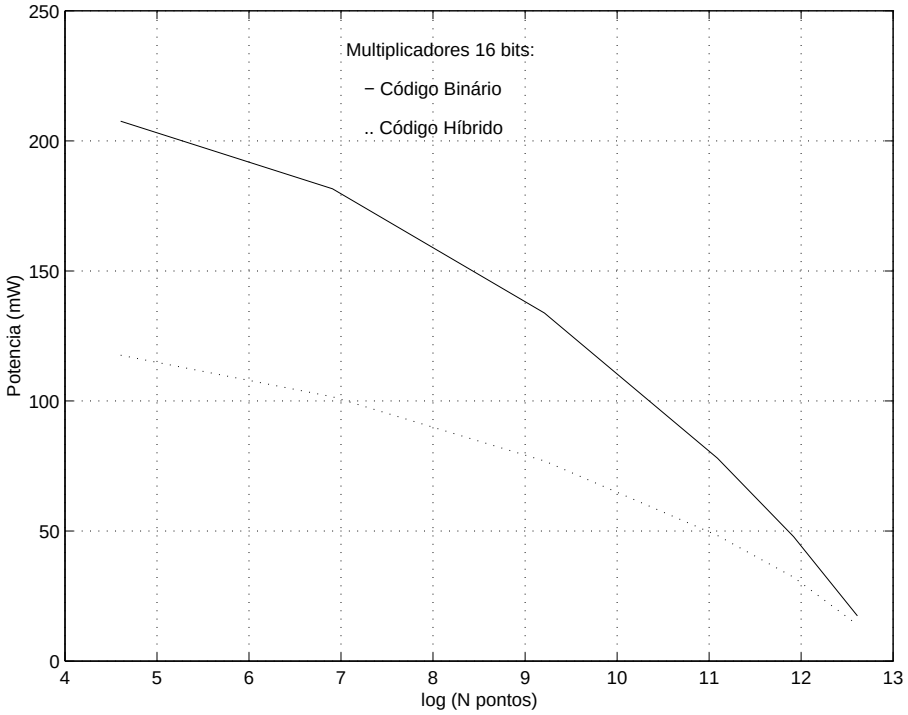

FIGURA 5.9 – Variação da potência com o número de pontos para multiplicadores de 16 bits.

Como pode ser observado na Figura 5.9, o maior número de pontos reduz progressivamente o consumo de potência dos circuitos. A partir da máxima representação do sinal (2^{16}), deve-se observar que o consumo de potência se reduz a níveis mínimos (próximos a zero). Isto devido à maior correlação entre os bits que determina um baixo número de transições no sinal. O aspecto da redução do consumo de potência para um maior número de pontos pode ser representativo para uma tendência à utilização de um número ainda maior de pontos. No entanto, nota-se que existe um determinado limite em que não se observa mais nenhum efeito de redução das variações entre dados consecutivos no sinal de entrada. Para este limite, não se torna mais possível praticamente nenhuma redução de consumo de potência dos circuitos.

5.2.4.1 Elementos Somadores e Multiplicadores Básicos

As arquiteturas de multiplicadores Binário e Híbrido na base 2^m são compostas por elementos básicos, que proporcionam as operações dos produtos parciais. De acordo

com o valor de m, são aplicados elementos somador e multiplicador básicos, cujas principais características são mostradas a seguir.

- **Elementos Somadores Básicos**

Os circuitos somadores básicos são responsáveis pelas somas dos produtos parciais. Além disto, estes circuitos estão presentes nos módulos dos termos dos produtos, como mostrado na Figura 5.7, para um multiplicador *array* Híbrido de 4 bits. Como visto na Seção 5.1.2.1, os circuitos somadores operando diretamente com o código Híbrido são mais complexos do que os somadores Binários e a sua melhor implementação para (m=2) é utilizar um somador Binário e fazer as conversões nas entradas e saídas. Um exemplo desta operação é mostrado na Figura 5.10 para um somador de 8 bits com m=2.

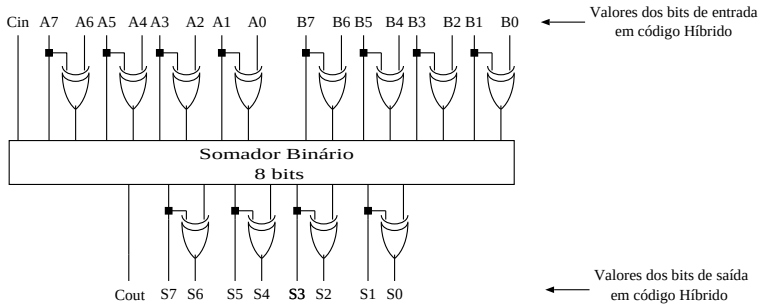

FIGURA 5.10 – Exemplo de um somador Híbrido de 8 bits com m=2.

Como visto no exemplo da Figura 5.10, para um circuito somador de 8 bits, com m=2, são necessárias 12 portas lógicas EXOR para conversão dos dados nas entradas e saídas. Para conversão de palavras de W bits nas entradas e saídas de um somador Híbrido com m=2, são necessárias $\frac{3}{2}W$ portas lógicas EXOR. Naturalmente, isto implica que os somadores Híbridos com m=2 serão mais complexos do que os somadores Binários apresentando um maior valor de área, como mostrado na Tabela 5.18.

Embora os somadores Híbridos com m=2 possam ser facilmente implementados utilizando somadores Binários e portas EXOR para a conversão dos dados de entrada e saída, esta mesma estratégia não pode ser adotada para grupos de bits m maiores do que 2. Isto devido à maior complexidade de conversão entre os códigos Binário e Gray para um número maior de bits (Equações 4.3 e 4.4). Desta forma, para grupos de bits m maiores do que 2, tem-se os circuitos somadores operando diretamente em código Híbrido (sem portas lógicas EXOR para conversão dos dados), observando-se um crescente aumento de complexidade destes operadores, como mostrado na Tabela 5.18.

A maior complexidade dos circuitos somadores para maiores valores de m se reflete nos valores de atraso. De fato, como pode ser observado na Tabela 5.18, com o aumento do valor de m, o atraso tende a aumentar para os somadores básicos Binário e Híbrido. Entretanto, devido à maior complexidade dos somadores Híbridos, estes circuitos apresentam sempre maiores valores de atraso em relação aos somadores Binários, como mostra a Tabela 5.18. Os maiores valores de atraso dos circuitos somadores Híbridos faz com que estes circuitos apresentem maiores valores de consumo de potência, como mostra a Tabela 5.19.

TABELA 5.18 – Valores de área e atraso para somadores básicos Binário e Híbrido.

Grupo de bits	Literais		Direrença (%)	Atraso(ns)		Direrença (%)
	Binário	Híbrido	Hib→Bin	Binário	Híbrido	Hib→Bin
m=2	28	40	+42,8	13,5	17,1	+26,6
m=4	80	426	+432,5	23,8	27,5	+15,5
m=8	2634	42210	+1502,5	44,9	161,5	+259,6

TABELA 5.19 – Valores de potência para somadores básicos Binário e Híbrido.

Grupo de bits	Potência (μW)					
	Seno		Diferença(%)	Randômico		Diferença(%)
	Bin	Hib	Hib→Bin	Bin	Hib	Hib→Bin
m=2	0,20	0,32	+60,0	220	380	+72,7
m=4	1,84	2,85	+54,8	564	1590	+181,9
m=8	302	2460	+714,5	11510	184650	+1504,0

Como pode ser observado na Tabela 5.19, para valores de entrada senoidais, onde o aspecto da correlação está presente, os circuitos somadores Binário e Híbrido apresentam menores valores de potência comparado aos valores com entradas randômicas, onde os valores são uniformemente distribuídos. O aspecto da falta de correlação nos sinais randômicos, faz com que os somadores Híbridos apresentem maiores valores de consumo de potência em relação aos circuitos Binários, comparado às entradas senoidais, como pode ser observado na Tabela 5.19.

- **Elementos Multiplicadores Básicos**

Na estrutura dos multiplicadores Binário e Híbrido na base 2^m, os módulos dos produtos são compostos por elementos multiplicadores básicos. Em um circuito multiplicador *array* de W bits são necessários m por m destes elementos. Os elementos básicos multiplicadores atuam diretamente nos códigos Binário e Híbrido, sendo que os elementos Híbridos não necessitam de nenhum *hardware* adicional para conversão dos dados de entrada e saída, como visto para os elementos somadores com m=2. Para valores de m=2 é possível implementar multiplicadores básicos Binário e Híbrido com mesmos valores de área e atraso, como mostra a Tabela 5.20. Entretanto, para os valores de m maiores do que 2, a complexidade do código Gray se torna mais expressiva nos grupos de bits e os elementos multiplicadores Híbridos apresentam maiores valores de área e atraso, como mostrado na Tabela 5.20. Entretanto, deve-se observar que as diferenças entre os elementos Binário e Híbrido não é tão significativa quanto nos elementos somadores. Isto devido à regularidade do multiplicador Híbrido básico, que permite a utilização do código Gray nos grupos de m bits, tendo-se o comportamento da lógica Binária entre os grupos (como mostrado na Tabela 5.7).

A maior complexidade apresentada pelos multiplicadores básicos para grupos de bits maiores do que 2, m=4 e 8, eleva o consumo de potência para os circuitos Binário e Híbrido, como mostrado na Tabela 5.21. Como pode ser visto nesta mesma tabela, o tipo de estrutura regular mostrada pelos multiplicadores básicos Híbridos permite a estes operadores apresentarem menores valores de consumo de potência

TABELA 5.20 – Valores de área e atraso para multiplicadores básicos Binário e Híbrido.

Grupo de bits	Literais		Direrença (%)	Atraso(ns)		Direrença (%)
	Binário	Híbrido	Hib→Bin	Binário	Híbrido	Hib→Bin
m=2	17	17	+0,0	4,7	4,7	0,0
m=4	532	751	+41,2	26,8	36,8	+37,3
m=8	161716	164372	+1,6	292,9	302,9	+34,1

em relação aos circuitos Binários, para vetores de entrada de *trace* real. Deve ser observado que quanto maior o valor de m maior o valor de potência das arquiteturas (devido à maior complexidade dos módulos básicos) e maior a redução de potência dos operadores Híbridos (devido à utilização do código Gray nos grupos de m bits do circuitos Híbrido, que apresenta uma menor atividade de transição, como mostrado na Tabela 5.8). Embora para as entradas de *trace* real (entradas senoidais), onde se tem uma maior correlação nos dados, os multiplicadores Híbridos básicos apresentam menores valores de potência, deve-se observar que para entradas uniformemente distribuídas, onde o aspecto da correlação não está presente, estes operadores apresentam maiores consumos de potência, como mostrado na Tabela 5.21, devido a sua maior complexidade.

TABELA 5.21 – Valores de potência para multiplicadores básicos Binário e Híbrido.

Grupo de bits	Potência (μW)					
	Seno		Diferença(%)	Randômico		Diferença(%)
	Bin	Hib	Hib→Bin	Bin	Hib	Hib→Bin
m=2	0,084	0,090	+7,0	118	119	+0,8
m=4	6,87	5,62	-18,1	2160	3170	+46,7
m=8	25600	12990	-49,2	888420	904980	+1,8

5.2.4.2 Multiplicação na Base 2^m

Na operação de multiplicação na base 2^m, os operandos são divididos em grupos de m bits. Cada um destes grupos pode ser visto como uma representação de um dígito em uma base 2^m. Desta forma, a arquitetura de um multiplicador na base 2^m segue a operação de multiplicação básica de números representados na base 2^m.

Considerando dois operandos de largura W bits, $A = \sum_{i=0}^{\frac{W}{m}-1} a_i 2^{i \cdot m}$ e $B = \sum_{j=0}^{\frac{W}{m}-1} b_j 2^{j \cdot m}$, onde cada a_i, b_j são dígitos de m bits na representação na base 2^m, tem-se:

$$A \times B = \sum_{j=0}^{\frac{W}{m}-1} A \cdot b_j 2^{j \cdot m} \tag{5.7}$$

onde se tem,

$$A \cdot b_j = \sum_{i=0}^{\frac{W}{m}-1} b_j \cdot a_i 2^{i \cdot m} \tag{5.8}$$

A Figura 5.11 ilustra esta operação para operadores W=8 bits na representação Binária utilizando a base 16 (m=4). Para o exemplo mostrado, os termos do produto parcial são obtidos pela multiplicação de cada grupo de m bits dos termos multiplicador e multiplicando. Desta forma, cada linha de produto parcial é processada por uma multiplicação $m \times W$, como mostrado na Figura 5.11(b).

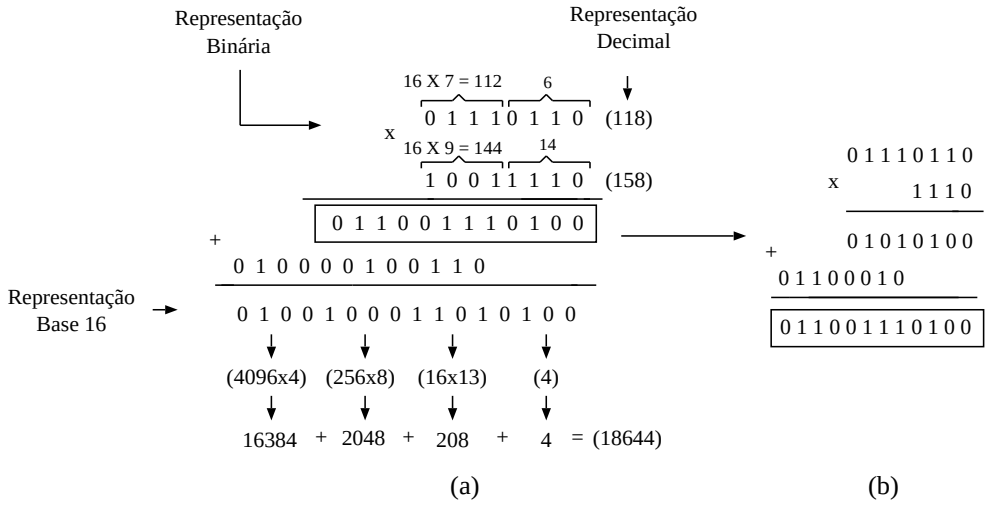

FIGURA 5.11 – Exemplo de uma multiplicação de 8 bits em código Binário na base 16.

A Figura 5.12 mostra o mesmo exemplo anterior para uma operação em código Híbrido. Como pode ser observado nesta figura, a operação de multiplicação neste código consiste na aplicação do código Gray nos grupos m de 4 bits.

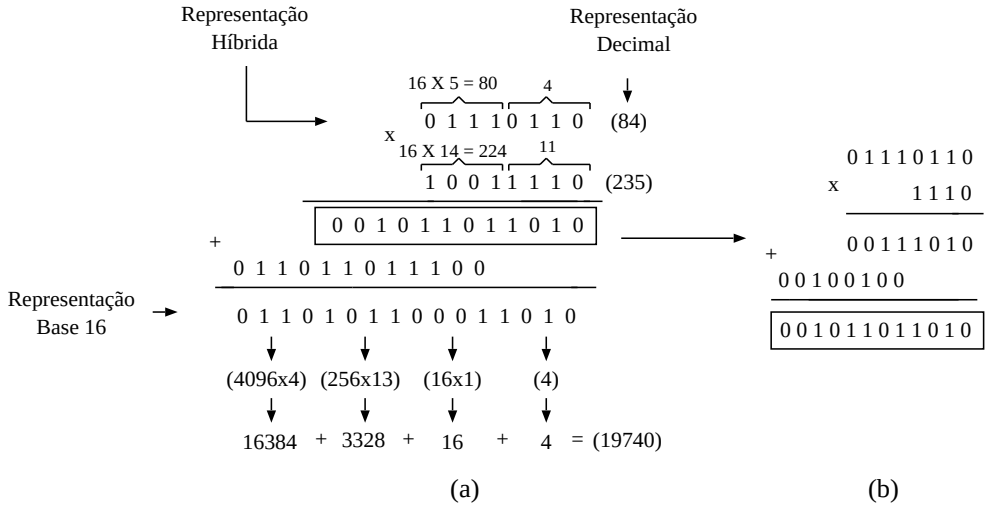

FIGURA 5.12 – Exemplo de uma multiplicação de 8 bits em código Híbrido na base 16.

5.2.4.3 Arquitetura do Multiplicador *Array* na Base 2^m

A estrutura da arquitetura do multiplicador *array* na base 2^m é a mesma do que um multiplicador tipo *array* convencional. Entretanto, cada linha do produto parcial opera em grupos de m bits ao invés de uma operação bit a bit. Isto reduz o número de linhas de produto para $\frac{W}{m}$. A Figura 5.13 mostra a arquitetura de um multiplicador *array* de 8 bits na base 16 (m=4). Deve-se observar que a arquitetura mostrada na Figura 5.13 é a mesma para operações em código Binário ou código Híbrido. A diferença consiste nos módulos básicos somador e multiplicador, que operam de acordo com os códigos correspondentes em suas entradas.

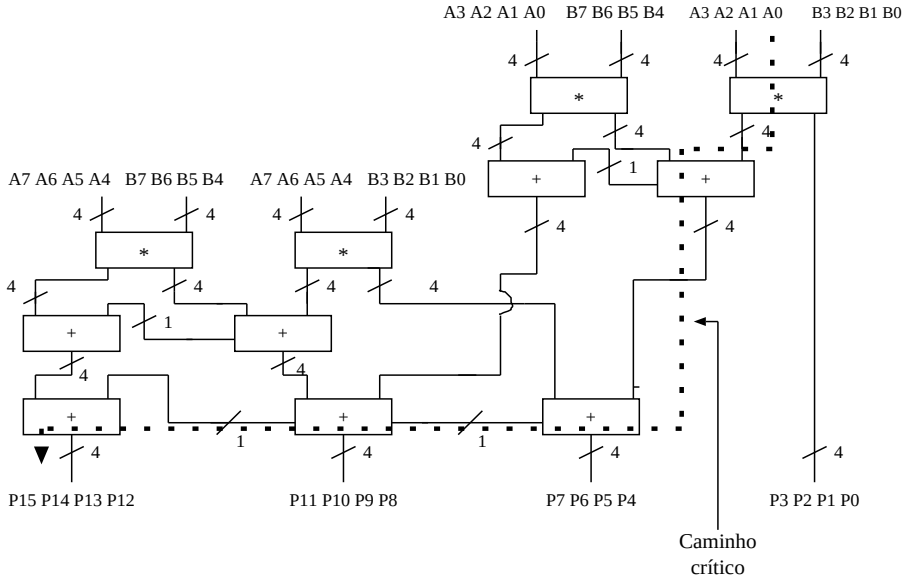

FIGURA 5.13 – Arquitetura de um multiplicador *array* de 8 bits na base 16.

Como pode ser observado na Figura 5.13, em um multiplicador de W bits são necessárias $\frac{W}{m}$ linhas de produtos, tendo cada uma $\frac{W}{m}$ módulos básicos de m por m multiplicadores e o mesmo número de m por m somadores. Uma linha adicional composta por $\frac{W}{m}+1$ desses elementos somadores básicos é responsável pela soma dos termos dos produtos parciais. Esta estrutura pode fornecer qualquer combinação de valores para W e m.

Nas arquiteturas dos multiplicadores *array* na base 2^m, pode ser observado o relacionamento entre a complexidade dos módulos básicos e a quantidade de linhas de produtos parciais. Quanto maior o valor de m, menor o número de linhas de produtos parciais e maior a complexidade dos módulos somador e multiplicador. Entretanto, como será mostrado na próxima seção, é possível gerar módulos otimizados para valores de m=2 e 4.

- **Caminho Crítico**

Os multiplicadores *array* na base 2^m são compostos na sua estrutura por elementos básicos somadores e multiplicadores. A complexidade das arquiteturas está associada ao grupo de m bits utilizado. O atraso de caminho crítico das arquiteturas varia de acordo com a quantidade e complexidade dos módulos básicos. A linha pontilhada da Figura 5.13 mostra o caminho crítico para a arquitetura de um

multiplicador *array* de 8 bits com m=4. Para o exemplo mostrado nesta figura, o caminho crítico é dado por um elemento multiplicador e um elemento somador do primeiro módulo do produto e pelos elementos somadores da última linha da soma dos produtos parciais.

Para uma arquitetura de W bits na base 2^m (com $\frac{W}{m}$>1), o número de elementos no caminho crítico é calculado pela expressão $\frac{3W}{m} - 1$. Esta expressão considera um elemento multiplicador e um elemento somador do primeiro módulo, dois elementos somadores de cada uma das $\frac{W}{m} - 2$ linhas de produtos parciais e $\frac{W}{m} + 1$ elementos somadores da última linha de produto parcial. Para uma muliplicação com $\frac{W}{m}$=1, o caminho crítico é dado pelo próprio elemento multiplicador básico.

5.2.4.4 Comparações de Desempenho

Embora as arquiteturas na base 2^m precisem de uma menor quantidade de elementos multiplicadores básicos em relação à arquitetura *array* convencional, deve-se observar que cada elemento multiplicador básico destas arquiteturas é composto por uma maior quantidade de portas lógicas. Desta forma, as novas arquiteturas de multiplicadores apresentam uma maior quantidade de portas lógicas para a realização dos produtos parciais em relação ao multiplicador *array* convencional. Logo, as novas arquiteturas de multiplicadores apre-sentam maiores valores de área em relação ao multiplicador *array* Binário convencional (m=1), para todos os tamanhos de grupos de bits, como mostrado na Tabela 5.22, assumindo W=16. Nesta tabela, os valores percentuais dos multiplicadores Binário e Híbrido com m=2, 4 e 8 representam comparações com o multiplicador Binário convencional com m=1. Esta tabela também mostra que a complexidade dos elementos básicos aumenta muito rapidamente com m. Além disto, deve-se observar que os multiplicadores *array* Híbridos apresentam maiores valores de área do que os multiplicadores Binários para todos os tamanhos de grupos de bits. Isto ocorre devido ao fato dos operadores somadores básicos Híbridos apresentarem maior valor de área em relação aos equivalentes operadores Binários, como mostrado na Tabela 5.18.

TABELA 5.22 – Valores de área para multiplicadores Binário e Híbrido para W=16.

Grupo de bits	Binário		Híbrido	
	Literais	%	Literais	%
m=1	3744	–	–	–
m=2	4314	+15,2	5017	+34,0
m=4	10537	+181,4	21311	+469,2
m=8	663474	+17621,0	765375	+20343,0

Apesar dos multiplicadores Binário e Híbrido apresentarem maiores valores de área, estas arquiteturas apresentam menores valores de atraso em relação ao multiplicador Binário com m=1, como mostra a Tabela 5.23. Isto se deve ao fato de que enquanto as arquiteturas Binária e Híbrida na base 2^m necessitam de $\frac{W}{m}-1$ linhas de somadores responsáveis pela soma dos termos dos produtos parciais, a arquitetura Binária com m=1 necessita de $W - 1$ linhas. Embora os m por m módulos sejam mais complexos, é possível reduzir o atraso de caminho crítico para os casos m=2 e m=4. Para o caso da arquitetura Binária com m=4, deve-se observar que uma

menor quantidade de linhas de produtos parciais necessárias (3 linhas) associado à menor quantidade de módulos básicos no caminho crítico, permite atingir um maior valor de redução de atraso entre as arquiteturas. O mesmo aspecto não é verificado para a arquitetura Híbrida com m=4. Isto devido à maior complexidade dos blocos básicos presentes no caminho crítico, que operam em código Gray. Para um maior valor de m, m=8, a complexidade dos módulos básicos aumenta consideravelmente de tal forma que os valores de atraso de ambas as arquiteturas aumentam significativamente, como mostra a Tabela 5.23.

Embora os elementos básicos somador e multiplicador Híbrido, com m=2, apresentados na seção anterior, possuam maiores de atraso em relação aos operadores Binários, deve-se observar na Tabela 5.23 que o multiplicador *array* Híbrido apresenta um menor valor de atraso, para este valor de m. Isto ocorre devido ao fato de termos introduzido no interior dos elementos básicos as portas lógicas EXOR, que são responsáveis pela conversão da representação dos dados. Esta otimização permitiu a redução da profundidade lógica e do atraso da arquitetura *array* Híbrida com m=2.

TABELA 5.23 – Valores de atraso para multiplicadores Binário e Híbrido para W=16.

Grupo de bits	Binário		Híbrido	
	Atraso(ns)	%	Atraso(ns)	%
m=1	260,0	–	–	–
m=2	212,5	-18,2	210,1	-19,1
m=4	177,1	-31,8	236,5	-9,0
m=8	403,7	+55,0	587,4	+126,0

Apesar da maior área apresentada pelos multiplicadores Binário e Híbrido base 2^m, para diferentes tamanhos de grupos de bits, estas arquiteturas consomem significativamente menos potência do que a arquitetura convencional Binária (m=1). A Tabela 5.24 mostra os resultados de potência entre o multiplicador *array* Binário convencional e as novas arquiteturas usando um sinal de *trace* real. Como pode ser observado nesta tabela, as arquiteturas Binária e Híbrida com m=2 podem reduzir potência em até quase 60%. Para m=4 o multiplicador Binário apresenta o menor valor de consumo de potência entre as arquiteturas. Este fato está relacionado ao significativo menor valor de atraso apresentado por esta arquitetura, como mostrado na Tabela 5.23. Para este mesmo grupo de bits (m=4), o multiplicador Híbrido apresenta um menor valor de redução de potência do que a arquitetura Binária, devido à maior complexidade dos elementos básicos. Entretanto, o valor de redução de potência desta arquitetura apresenta um consumo de potência quase 55% menor em relação ao multiplicador *array* convencional, devido ao menor número de linhas de produtos parciais. Como pode ser observado na Tabela 5.24, para m=8, as novas arquiteturas apresentam maiores valores de consumo de potência. Isto devido a grande complexidade dos elementos básicos somador e multiplicador que compõem estas arquiteturas.

Como pode ser observado na Tabela 5.25, para sinais de padrões randômicos nas entradas dos multiplicadores, onde o aspecto da correlação do sinal não se faz presente, resultados similares são obtidos, ou seja, reduções de potência acima dos 50% são obtidas nos multiplicadores Binários m=2,4 e no multiplicador Híbrido

TABELA 5.24 – Potência para multiplicadores Binário e Híbrido com entradas *trace* real para W=16.

Grupo de bits	Binário		Híbrido	
	Potência(mW)	%	Potência(mW)	%
m=1	133,80	–	–	–
m=2	55,72	-58,3	55,89	-58,2
m=4	35,44	-73.5	59,47	-55,5
m=8	1321,00	+887,3	1818,60	+1259,2

m=2. Naturalmente, os resultados tendem a piorar para as arquiteturas com maiores valores de m devido à maior complexidade dos elementos básicos.

TABELA 5.25 – Potência para multiplicadores Binário e Híbrido com entradas randômicas para W=16.

Grupo de bits	Binário		Híbrido	
	Potência(mW)	%	Potência(mW)	%
m=1	190,77	–	–	–
m=2	81,48	-57,2	86,44	-54,7
m=4	66,74	-53,3	125,13	-34,4
m=8	3156,00	+1554,3	4226,00	+2115,2

5.2.4.5 Carga Capacitiva no Barramento de Dados

Neste trabalho, investiga-se o impacto do consumo de potência das arquiteturas Binária e Híbrida, com m=2, usando diferentes cargas capacitivas para as entradas das linhas dos barramentos. Os resultados obtidos estão indicados na Tabela 5.26, para W=16, onde a carga capacitiva é indicada em termos de múltiplos de capacitância de entrada equivalente de circuitos *buffers* de tamanho mínimo.

TABELA 5.26 – Resultados de potência para diferentes cargas capacitivas de entrada, m=2 e W=16.

Carga capacitiva	Binário	Híbrido	Dif. (Híbrido→Binário)
5	57,31mW	56,94mW	-0,6%
50	71,76mW	69,05mW	-3,7%
100	87,81mW	82,5mW	-6,0%
200	119,91mW	109,4mW	-8,7%

Como pode ser visto na Tabela 5.26, para elevados valores de cargas capacitivas, o circuito Híbrido consome progressivamente menor valor de potência em relação ao circuito Binário. Isto ocorre devido ao fato do sinal de entrada *trace* real em código Híbrido apresentar menos transições do que o equivalente sinal Binário, reduzindo desta forma a capacitância chaveada nos barramentos de entrada. De fato, tem-se observado que o sinal de *trace* real Híbrido reduz a potência acima de 16%

em relação ao mesmo sinal em código Binário. Desta forma, na presença de elevados valores de carga capacitiva nos barramentos de entrada de dados, e para o caso do projetista ter a flexibilidade de escolher os operandos em um código diferente, o circuito multiplicador em código Híbrido pode ser a melhor escolha para a redução de potência.

5.3 Ordenação e Particionamento de Coeficientes

Como visto no Capítulo 4, a ordenação dos sinais de entrada é uma das principais técnicas de redução da atividade de chaveamento em circuitos digitais. Este aspecto está relacionado com a correlação dos sinais, dependendo da faixa dinâmica de operação dos dados. Em circuitos da área de processamento digital de sinais esta técnica se torna atrativa, devido às características dos circuitos que envolvem uma grande quantidade de operadores de multiplicação e soma. Em particular, a técnica de manipulação de coeficientes visando a sua melhor ordenação foi apresentada em alguns trabalhos de pesquisa [MEH 95, MEH 96, MEH 98]. Nesta seção, propõe-se a extensão desta técnica, tendo-se além da ordenação um melhor particionamento dos coeficientes para a redução da distância de *Hamming* entre palavras sucessivas. Além disto, discute-se a utilização destas técnicas no algoritmo FFT. Nesta parte do trabalho será mostrado o impacto da aplicação das técnicas de ordenação e particionamento de coeficientes em arquiteturas seqüenciais e semi-paralelas dos algoritmos de filtro FIR e FFT. Estas arquiteturas serão mostradas em detalhe no Capítulo 7. Os resultados de potência das arquiteturas são apresentados em potência normalizada, que representa a quantidade de energia gasta em um ciclo de relógio.

5.3.1 Manipulação de Coeficientes em Algoritmos de Filtros FIR

Em [MEH 95, MEH 96, MEH 98], propõe-se um algoritmo de ordenação dos coeficientes em filtros FIR. Dado que a operação em filtros FIR é comutativa e associativa, a saída dos filtros FIR é independente da ordem de computação dos produtos dos coeficientes. Desta forma, tem-se que para um filtro FIR de ordem 4 por exemplo, a saída pode ser computada de acordo com a Equação 5.9 ou com a Equação 5.10 [MEH 95, MEH 96, MEH 98].

$$Y_n = A_0X_n + A_1X_{n-1} + A_2X_{n-2} + A_3X_{n-3} \tag{5.9}$$

$$Y_n = A_1X_{n-1} + A_3X_{n-3} + A_0X_n + A_2X_{n-2} \tag{5.10}$$

A ordem da computação dos produtos dos dados de entrada com os coeficientes afeta diretamente a seqüência com que os coeficientes aparecem no barramento de dados. Esta computação também influencia a ordem dos dados, mas sobre esses não temos controle. Desta forma, o algoritmo proposto em [MEH 95, MEH 96, MEH 98] busca encontrar um ordenamento ótimo dos coeficientes de tal forma que a distância de *Hamming* entre coeficientes consecutivos possa ser minimizada. Para um filtro de ordem N, pode-se ter $N!$ possíveis diferentes ordenamentos dos coeficientes. Uma vez que a seqüência de acesso aos coeficientes é identificada, os coeficientes e os correspondentes valores de dados podem ser reordenados e apropriadamente armazenados na memória, tal que a seqüência desejada das computações dos produtos

dados-coeficientes é encontrada quando a memória é acessada seqüencialmente.

Como pode ser observado, o algoritmo de ordenamento proposto em [MEH 95, MEH 96, MEH 98] considera uma seqüência de coeficientes para um circuito totalmente seqüencial de tal forma que o melhor ordenamento possa ser encontrado.

No presente trabalho de pesquisa, experimenta-se uma extensão desta técnica em uma arquitetura de filtro FIR semi-paralela de 8ª ordem, onde o *hardware* é duplicado e os coeficientes são particionados em dois grupos de quatro coeficientes. Desta forma, o problema é direcionado a encontrar o melhor particionamento para cada coeficiente a partir do cálculo da mínima distância de *Hamming* entre os coeficientes em cada grupo, como mostra o pseudo-código da Figura 5.14.

```
1.  for all permutations of coefficients H(0-7){
2.     partition1=Hamming((H[0],H[1]) + (H[1],H[2]) + (H[2],H[3]) + (H[3],H[0]));
3.     partition2=Hamming((H[4],H[5]) + (H[5],H[6]) + (H[6],H[7]) + (H[7],H[0]));
4.     cost function = partition1 + partition2;
5.        if cost function < minimum found{
6.           save current partition;
7.           minimum = cost function;
8. 9.     }
10. }
```

FIGURA 5.14 – Algoritmo para a geração de particionamento e ordenação de coeficientes.

Como mostrado no pseudo-código da Figura 5.14, a função de custo é calculada pela computação de todas as combinações em um método exaustivo. Deste modo, a melhor função de custo pode ser encontrada de uma forma rápida para um pequeno número de coeficientes. Neste trabalho, para 8 coeficientes o número de permutações é 8!=40320 que é ainda razoável. Entretanto, como pode ser observado se um elevado número de coeficientes é utilizado, este algoritmo é menos atrativo devido ao tempo necessário para processar o elevado número de combinações possíveis. Neste caso, deve-se utilizar um algoritmo diferente baseado em heurísticas no sentido de obtenção da solução ótima.

5.3.1.1 Resultados da Aplicação da Manipulação de Coeficientes

A técnica de ordenação dos coeficientes proposta em [MEH 95, MEH 98a, MEH 98], parte do princípio de que os coeficientes na arquitetura de um filtro FIR seqüencial podem ser reagrupados de forma a reduzir a atividade de chaveamento na entrada do circuito multiplicador. Isto devido ao fato da operação de soma no algoritmo do filtro neste tipo de arquitetura ser comutativa e associativa e desta forma, a saída do filtro é independente da ordem de processamento do produto dos coeficientes. A Tabela 5.27 mostra os resultados de consumo de potência para a arquitetura seqüencial com operadores aritméticos em códigos Binário e Híbrido utilizando o algoritmo proposto em [MEH 95, MEH 98a, MEH 98] para a melhor ordenação dos coeficientes.

Para o grupo de coeficientes utilizado nos exemplos das arquiteturas de filtro FIR, observa-se que a técnica de ordenação dos coeficientes não apresenta uma redução significativa do consumo de potência. Para este grupo, a técnica de ordenação se mostra mais eficiente para a arquitetura com operadores em código

TABELA 5.27 – Valores de potência para arquiteturas de filtro FIR seqüenciais com ordenação de coeficientes.

Operadores	Grupo de Bits	Coeficientes Originais	Coeficientes com Ordenação	Diferença Ordenação vs. Originais
Binários	m=2	155,7	156,5	+0,8
	m=4	128,9	129,1	+0,2
Híbridos	m=2	180,7	176,9	-3,8
	m=4	228,5	216,0	-12,5

Híbrido, como mostra a Tabela 5.27. Isto devido à menor distância de *Hamming* verificada nos coeficientes com esta representação, após a utilização da técnica de ordenação dos coeficientes. A melhor ordenação dos coeficentes em conjunto com a maior correlação dos dados de entrada nos multiplicadores contribui para uma maior redução da potência na arquitetura com código Híbrido.

Embora para o grupo de coeficientes utilizado como exemplo a técnica de ordenação de coeficientes não tenha apresentado valores significativos de redução de potência, esta técnica pode se tornar efetiva em grupos de coeficientes que apresentem uma maior correlação, principalmente nos operadores com código Híbrido. Entretanto, a alternativa seqüencial que utiliza esta técnica possui uma restrição de operação em 8 ciclos de relógio, para os filtros utilizados como exemplo neste trabalho, para processamento de cada amostra comprometendo desta forma, a velocidade de operação do circuito. De outro modo, a alternativa arquitetural semi-paralela permite a operação das arquiteturas do filtro FIR no dobro da freqüência de relógio em relação às arquiteturas seqüenciais. Desta forma, torna-se interessante a aplicação neste tipo de alternativa arquitetural, do algoritmo de ordenação e particionamento de coeficientes. A Tabela 5.28 mostra os resultados de potência para alternativa semi-paralela após aplicação do algoritmo proposto para ordenação e particionamento dos coeficientes.

TABELA 5.28 – Valores de potência para arquiteturas de filtro FIR semi-paralelas com ordenação e particionamento de coeficientes.

Operadores	Grupo de Bits	Coeficientes Originais	Coeficientes com Ord. e Particionamento	Diferença Ordenação vs. Orig.
Binários	m=2	136,9	132,5	-4,3
	m=4	112,8	110,8	-2,1
Híbridos	m=2	158,3	151,5	-6,7
	m=4	215,0	201,0	-13,9

Como pode ser observado na Tabela 5.28, a técnica de ordenação e particionamento apresenta uma maior redução de potência para o grupo de coeficientes utilizado. Observa-se que da mesma forma como mostrado para a arquitetura seqüencial, a aplicação da técnica de ordenação e particionamento se mostra mais eficiente com o-peradores em código Híbrido para o grupo de coeficientes utilizado. Deve-se observar que o aumento do *hardware*, em particular dos circuitos multiplicadores,

TABELA 5.29 – Valores de potência para arquiteturas seqüenciais e semi-paralelas após manipulação de coeficientes e redução de V_{dd}.

	Grupo de Bits	Potência Seqüencial (com Ordenação)	Potência Semi-Paralela com $V_{dd}/2$ (com Ord. e Part.)	Diferença Semi-Par vs. Seq
Bin	m=2	156,5	132,5/4 = 33,1	-123,4
	m=4	129,1	110,8/4 = 27,6	-101,4
Hib	m=2	176,9	151,5/4 = 37.8	-139,0
	m=4	216,0	201,7/4 = 50,2	-165,7

contribui para uma maior correlação da multiplicação dos dados pelos coeficientes, o que leva a uma maior redução da potência nestas arquiteturas em relação às arquiteturas seqüenciais, como pode ser comparado nas Tabelas 5.27 e 5.28. Outro aspecto a ser destacado é que as alternativas seqüencial e semi-paralela com operadores em código Híbrido com m=4 apresentam as maiores reduções de potência. Isto se deve ao fato dos coeficientes em código Híbrido com este valor de m apresentarem uma menor quantidade de transições e após a manipulação dos coeficientes, tem-se uma menor distância de *Hamming* nestes coeficientes.

Além do aspecto da maior redução da potência pela utilização da técnica de ordenação e particionamento dos coeficientes, observa-se que as arquiteturas semi-paralelas podem ter uma redução da sua potência por um fator de 4, pela redução de V_{dd} pela metade, para manutenção do mesmo *throughput* em relação às arquiteturas seqüenciais, como mostram os resultados da Tabela 5.29.

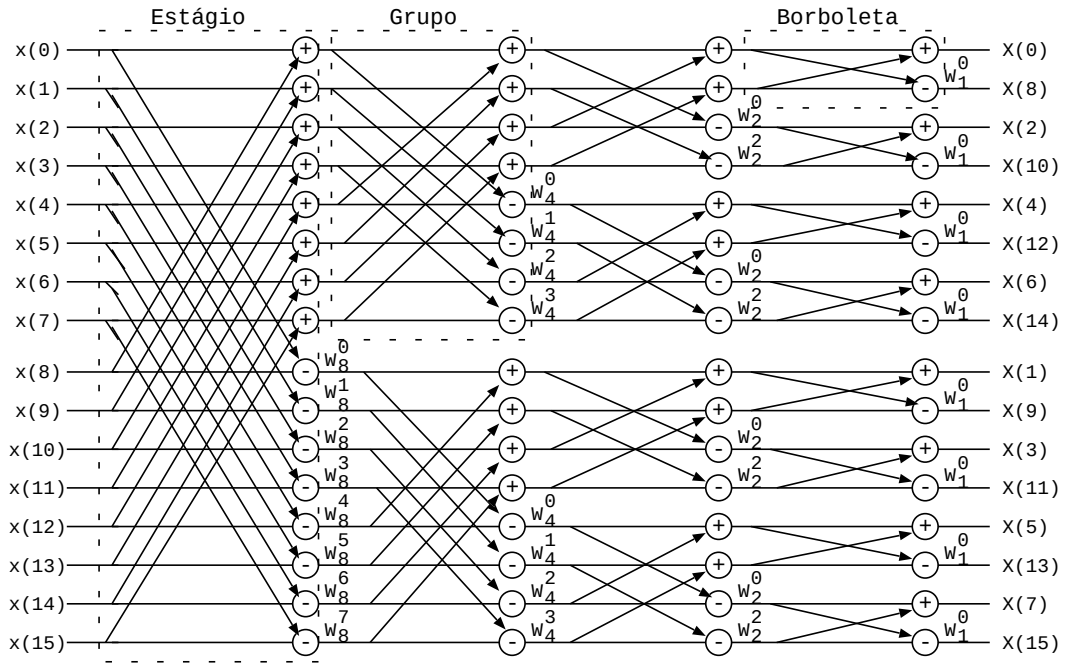

FIGURA 5.15 – Fluxo de dados de uma FFT de fator comum na base 2 com decimação em freqüência de 16 pontos.

5.3.2 Manipulação de Coeficientes em Algoritmo de Transformada Rápida de Fourier

A transformada rápida de Fourier (FFT) é uma classe de algoritmos eficientes para processamento da transformada discreta de Fourier (DFT). Os algoritmos FFT são projetados tipicamente para minimizar o número de multiplicações e adições, mantendo-se uma forma simples em sua estrutura.

Existe uma série de algoritmos para o cálculo da FFT [OPP 89]. A classe mais popular destes algoritmos é chamada FFT de fator comum. Estes algoritmos também são denominados Cooley-Tukey por terem sido popularizados por estes pesquisadores [COO 65]. Entre os algoritmos de fator comum os de base 2 e base 4 apresentam maior simplicidade nas suas estruturas computacionais. Os algoritmos de fator comum com bases maiores de que 4 apresentam uma estrutura mais complexa com reduzida eficiência computacional [BAA 2000].

No nosso trabalho, o principal enfoque é o algoritmo de fator comum na base 2 com decimação em freqüência. Neste algoritmo os valores das amostras em freqüência são decimados durante cada estágio da FFT. As operações são efetuadas em um par de sinais de entrada. A estrutura básica de um algoritmo de FFT na base 2 com decimação em freqüencia de 16 pontos é mostrada na Figura 5.15.

Como pode ser observado pela Figura 5.15, a estrutura computacional de uma FFT é dividida em estágios, grupos e borboletas. O cálculo da quantidade de estágios de uma FFT de fator comum na base 2 é dado por $log_2 N$, onde N representa o número de pontos da FFT.

Na estrutura mostrada na Figura 5.15, uma borboleta representa a parte central de cálculo da FFT. A complexidade da borboleta está associada ao tipo de algoritmo utilizado. A Figura 5.16 mostra a estrutura padrão de uma borboleta para o cálculo do algoritmo de FFT com decimação em frequência de fator comum na base 2 utilizado neste trabalho.

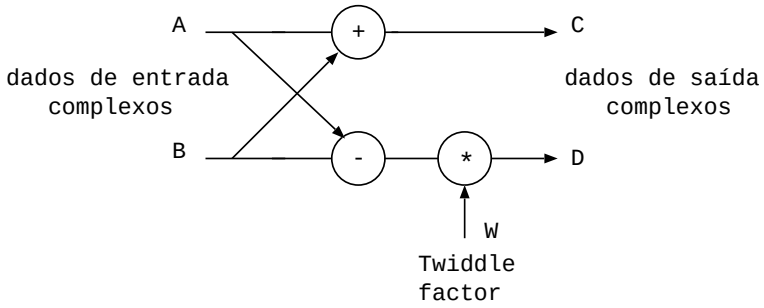

FIGURA 5.16 – Diagrama de uma borboleta com decimação em freqüência de fator comum na base 2.

Na estrutura da borboleta mostrada na Figura 5.16, as amostras são multiplicadas por coeficientes fixos (*twiddle factor*). Estes coeficientes representam valores múltiplos de um fator $\frac{2\pi}{N}$, onde N é o número de pontos da FFT. Em uma implementação do algoritmo FFT, os coeficientes são armazenados seqüencialmente em localizações de memória e são lidos a cada processamento da borboleta. Entretanto, em cada estágio da FFT o resultado de cálculo da borboleta é independente da ordem de computação dos produtos dos coeficientes. Desta forma, os coeficientes podem ser reagrupados de forma a minimizar a distância de *Hamming* entre palavras

sucessivas. A Tabela 5.30 mostra um exemplo da seqüência de cálculos da primeira coluna da FFT, mostrada na Figura 5.15, considerando uma estrutura totalmente seqüencial com as operações aritméticas normais e após aplicação do algoritmo de ordenação de coeficientes. Como deve ser observado nesta mesma tabela, em uma estrutura totalmente seqüencial são necessários 8 ciclos de relógio para o cálculo de um estágio de uma FFT de 16 pontos de fator comum na base 2.

TABELA 5.30 – Ordenação de coeficientes no algoritmo FFT.

Ciclos de Relógio	Operação Normal	Operação com Ordenação
ciclo 1	$x_0 + x_8$ $(x_0 - x_8) \times W_8^0$	$x_0 + x_8$ $(x_0 - x_8) \times W_8^0$
ciclo 2	$x_1 + x_9$ $(x_1 - x_9) \times W_8^1$	$x_6 + x_{14}$ $(x_6 - x_{14}) \times W_8^6$
ciclo 3	$x_2 + x_{10}$ $(x_2 - x_{10}) \times W_8^2$	$x_2 + x_{10}$ $(x_2 - x_{10}) \times W_8^2$
ciclo 4	$x_3 + x_{11}$ $(x_3 - x_{11}) \times W_8^3$	$x_5 + x_{13}$ $(x_5 - x_{13}) \times W_8^5$
ciclo 5	$x_4 + x_{12}$ $(x_4 - x_{12}) \times W_8^4$	$x_3 + x_{11}$ $(x_3 - x_{11}) \times W_8^3$
ciclo 6	$x_5 + x_{13}$ $(x_5 - x_{13}) \times W_8^5$	$x_1 + x_9$ $(x_1 - x_9) \times W_8^1$
ciclo 7	$x_6 + x_{14}$ $(x_6 - x_{14}) \times W_8^6$	$x_4 + x_{12}$ $(x_4 - x_{12}) \times W_8^4$
ciclo 8	$x_7 + x_{15}$ $(x_7 - x_{15}) \times W_8^7$	$x_7 + x_{15}$ $(x_7 - x_{15}) \times W_8^7$

Pelo fato dos resultados do processamento da borboleta serem armazenados nas mesmas localizações de memória originais para serem utilizados nos cálculos do estágio posterior, deve-se observar que a nova ordenação dos coeficientes deve ser realizada em cada estágio da FFT. Este aspecto se torna atrativo, pois se tem um maior grau de liberdade para a exploração da melhor forma de acesso aos coeficientes em cada estágio da FFT. Entretanto, deve-se também observar que quanto maior o número de pontos utilizado na FFT, maior o tamanho da memória de leitura (*Read Only Memory - ROM*) necessária para o armazenamento dos coeficientes [CHA 99, MA 99, HAS 2002]. Para uma arquitetura totalmente seqüencial o tamanho desta memória é $N/2$.

Em uma arquitetura FFT semi-paralela, duas borboletas podem ser utilizadas simultaneamente para processamento das amostras. Desta forma, o tamanho da memória ROM para armazenamento dos coeficientes pode ser reduzido para $N/4$. Neste caso, o problema é direcionado a encontrar a melhor ordenação a partir do cálculo da mínima distância de *Hamming* em cada grupo de $N/4$ coeficientes particionados. A Tabela 5.31 mostra um exemplo da melhor ordenação dos coeficientes particionados da primeira coluna da FFT (Figura 5.15). Além da possibilidade de redução da atividade de transição nas entradas dos multiplicadores, pela redução da distância de *Hamming* entre coeficientes consecutivos em cada grupo da FFT, a

TABELA 5.31 – Ordenação dos coeficientes particionados no algoritmo FFT.

Ciclos de Relógio	Operação Normal		Operação com Ordenação e Particionamento	
ciclo 1	$x_0 + x_8$ $(x_0 - x_8) \times W_8^0$	$x_1 + x_9$ $(x_1 - x_9) \times W_8^1$	$x_0 + x_8$ $(x_0 - x_8) \times W_8^0$	$x_5 + x_{13}$ $(x_5 - x_{13}) \times W_8^5$
ciclo 2	$x_2 + x_{10}$ $(x_2 - x_{10}) \times W_8^2$	$x_3 + x_{11}$ $(x_3 - x_{11}) \times W_8^3$	$x_6 + x_{14}$ $(x_6 - x_{14}) \times W_8^6$	$x_1 + x_9$ $(x_1 - x_9) \times W_8^1$
ciclo 3	$x_4 + x_{12}$ $(x_4 - x_{12}) \times W_8^4$	$x_5 + x_{13}$ $(x_5 - x_{13}) \times W_8^5$	$x_2 + x_{10}$ $(x_2 - x_{10}) \times W_8^2$	$x_3 + x_{11}$ $(x_3 - x_{11}) \times W_8^3$
ciclo 4	$x_6 + x_{14}$ $(x_6 - x_{14}) \times W_8^6$	$x_7 + x_{15}$ $(x_7 - x_{15}) \times W_8^7$	$x_4 + x_{12}$ $(x_4 - x_{12}) \times W_8^4$	$x_7 + x_{15}$ $(x_7 - x_{15}) \times W_8^7$

alternativa semi-paralela permite a operação da FFT na metade dos ciclos de relógio que são utilizados em uma arquitetura totalmente seqüencial.

5.3.2.1 Resultados da Aplicação da Manipulação de Coeficientes

Em um algoritmo FFT os coeficientes (*twiddle factors*) representam valores múltiplos de um fator $\frac{2\pi}{N}$, onde N é o número de pontos da FFT. Estes coeficientes são armazenados seqüencialmente em localizações de memória e lidos a cada processamento da FFT. Em cada estágio da FFT são realizadas operações nas borboletas que envolvem a multiplicação de dados pelos coeficientes apropriados. A ordem com que os dados são processados em um estágio da FFT são independentes da ordem de processamento do produto dos coeficientes, visto que o resultado da operação só é utilizado no estágio seguinte. Desta forma, os coeficientes podem ser reagrupados em cada estágio da FFT de forma que a distâncias de *Hamming* entre palavras consecutivas possa ser minimizada. Com esta finalidade, utiliza-se o algoritmo de ordenação de coeficientes para aplicação em cada estágio do algoritmo FFT. Para uma arquitetura totalmente seqüencial de 16 pontos, realiza-se uma ordenação de 8 coeficentes reais e 8 coeficientes imaginários em cada estágio. A Tabela 5.32 mostra os resultados de consumo de potência para uma arquitetura totalmente seqüencial de 16 pontos com operadores aritméticos em códigos Binário e Híbrido.

TABELA 5.32 – Valores de potência para arquiteturas seqüenciais com ordenação de coeficientes.

Operadores	Grupo de Bits	Coeficientes Originais	Coeficientes com Ordenação	Diferença Ordenação vs. Originais
Binários	m=2	96,9	85,4	-11,5
	m=4	84,9	78,2	-6,7
Híbridos	m=2	110,8	91,7	-19,1
	m=4	101,1	91,4	-9,7

O grau de oportunidade para a aplicação da ordenação dos coeficentes a cada estágio da FFT faz com que a técnica de ordenação de coeficientes se mostre eficiente nas arquiteturas FFT. Como deve ser observado na Tabela 5.32, para as arquiteturas

com operadores aritméticos em códigos Binário e Híbrido com valores de m=2 e 4, há uma redução significativa do consumo de potência. Em particular, nas arquiteturas com operadores aritméticos com valor de m=2, tem-se uma maior eficiência do algoritmo de ordenação de coeficientes. Nas aquiteturas com operadores aritméticos em código Híbrido esta redução se torna mais significativa. Isto se deve ao fato da menor quantidade de transições apresentada nos coeficientes em código Híbrido pela maior correlação entre os bits.

Nas arquiteturas semi-paralelas, tem-se o particionamento dos coeficientes em $N/4$ grupos em cada estágio. A cada ciclo de relógio são utilizados quatro coeficientes simultaneamente no processamento de duas amostras reais e duas amostras imaginárias. A ordem com que os dados são processados nos $N/4$ grupos em cada estágio da FFT são independentes da ordem de processamento do produto dos coeficientes. Desta forma, a aplicação do algoritmo de ordenação de coeficientes na arquitetura semi-paralela consiste em reagrupar os coeficentes internamente em cada um dos $N/4$ grupos de coeficientes de forma que a distância de *Hamming* possa ser minimizada em cada um destes grupos. Os coeficientes são particionados da forma mostrada na Tabela 5.31. A aplicação da técnica em uma arquitetura semi-paralela de 16 pontos envolve então a aplicação da ordenação em dois grupos de 4 coeficientes reais e dois grupos de 4 coeficientes imaginários em cada estágio da FFT. A Tabela 5.33 mostra os resultados de potência para as arquiteturas semi-paralelas com operadores aritméticos em códigos Binário e Híbrido.

TABELA 5.33 – Valores de potência para arquiteturas semi-paralelas com ordenação em coeficientes particionados.

Operadores	Grupo de Bits	Coeficientes Originais	Coeficientes Particionados	Diferença Particionamento vs. Originais
Binários	m=2	82,1	66,4	-15,7
	m=4	72,8	57,9	-14,9
Híbridos	m=2	93,7	75,5	-18,2
	m=4	87,7	70,1	-17,6

TABELA 5.34 – Valores de potência para arquiteturas seqüenciais e semi-paralelas após manipulação de coeficientes e redução de V_{dd}.

	Grupo de Bits	Potência Seqüencial (com Ordenação)	Potência Semi-Par com $V_{dd}/2$ (com Coef. Particionados)	Diferença Semi-Par vs. Seq
Bin	m=2	85,4	66,4/4 = 16,6	-68,8
	m=4	78,2	57,9/4 = 14,5	-63,7
Hib	m=2	91,7	75,5/4 = 18,9	-72,8
	m=4	91,4	70,1/4 = 17,5	-73,9

Assim como nas arquiteturas seqüenciais, observa-se que nas arquiteturas semi-paralelas as maiores reduções de potência são verificadas nas arquiteturas com operadores em código Híbrido, pelo aspecto da menor quantidade de transições nos

coeficientes representados por esta codificação. O aspecto da aplicação da técnica de ordenação em um menor grupo de coeficientes particionados faz com que aumente a proximidade entre os valores dos coeficientes. Este aspecto faz com que aumente a correlação entre os bits dos coeficientes, tendo-se desta forma, as arquiteturas com os valores de m=2 e 4 com praticamente os mesmos valores de redução de potência, como pode ser observado na Tabela 5.33. Outro aspecto a ser destacado é o maior impacto em redução de potência nas arquiteturas semi-paralelas pela aplicação da técnica de ordenação nos coeficientes particionados. Nestas arquiteturas, apresenta-se até 26% de redução de potência em relação às arquiteturas seqüenciais com ordenação dos coeficientes, como pode ser comparado nas Tabelas 5.32 e 5.33.

Como foi observado nas Tabelas 5.32 e 5.33, a técnica de ordenação de coeficientes se torna atrativa no algoritmo FFT para a redução de potência nas arquiteturas seqüencial e semi-paralela. Em particular, nas alternativas semi-paralelas, a aplicação da técnica de ordenação de coeficientes faz com que estas arquiteturas apresentem os menores valores de potência. Além disto, observa-se que as arquiteturas semi-paralelas podem apresentar ainda uma redução em seus valores de potência por um fator de 4, pela redução de V_{dd} pela metade, para manutenção do mesmo *thoughput* em relação às arquiteturas seqüenciais. Isto pelo fato das arquiteturas semi-paralelas operarem na metade dos ciclos de relógio das arquiteturas seqüenciais. A Tabela 5.34 mostra os valores de potência, após a manipulação de coeficientes e redução da tensão de alimentação, onde se nota a grande redução de potência que pode ser proporcionada pelas arquiteturas semi-paralelas nestas condições.

5.4 Resumo

Este capítulo apresentou as técnicas de redução de potência a partir da redução da atividade de transição em barramentos de dados e operadores aritméticos. Como abordagem inicial, foram apresentadas técnicas de codificação aplicadas a operadores aritméticos. Foi observado que a implementação de operadores aritméticos operando diretamente com diferentes entradas codificadas pode apresentar uma grande complexidade de *hardware* com penalidade no consumo de potência. Entretanto, foram apresentadas implementações de operadores aritméticos regulares operando diretamente em um código Híbrido, que é um compromisso entre a mínima dependência das entradas apresentada pelo código Binário e a característica de baixa atividade de chaveamento apresentada pelo código Gray. Neste contexto, foram apresentados multiplicadores tipo *array* operando diretamente em código Híbrido com aspectos de redução de atraso e consumo de potência. Como pôde ser verificado, apesar das arquiteturas de multiplicadores em código Híbrido apresentarem uma penalidade em área de mais de 30% em relação aos operadores em código Binário convencional (m=1), estas arquiteturas podem apresentar uma redução de mais de 15% em atraso e mais de 50% em consumo de potência considerando o efeito de *glitching*. Isto devido ao aspecto da correlação dos sinais que contribui para a redução da atividade de chaveamento nestes operadores. Entretanto, foi observado que a arquitetura *array* utilizada para operar em código Híbrido, com redução das linhas de produtos parciais, pode também ser utilizada para operação em código Binário. Neste caso, tem-se valores mais próximos de atraso e consumo de potência entre as duas arquiteturas. Uma outra técnica para redução de potência apresentada neste capítulo, refere-se

à redução da atividade de chaveamento em operações dos algoritmos de filtro FIR e FFT. Em particular, foi abordada uma técnica de manipulação de coeficientes para redução da atividade de chaveamento na entrada dos circuitos multiplicadores a partir de um algoritmo proposto em [MEH 95, MEH 96, MEH 98]. Este algoritmo tem por objetivo encontrar a melhor ordenação dos coeficientes para aplicação em arquiteturas seqüenciais. Neste capítulo, foi apresentada uma extensão desta técnica com a proposta de um algoritmo para busca da melhor ordenação e do melhor particionamento dos coeficientes para aplicação em arquiteturas semi-paralelas de filtro FIR. Para as arquiteturas FFT foi utilizado o algoritmo de ordenação de coeficientes em arquiteturas seqüenciais e semi-paralelas. Como foi observado, o maior grau de oportunidade para ordenação de coeficientes nos estágios da FFT permitiu maiores reduções de potência neste algoritmo. Em particular, a arquitetura semi-paralela se mostrou mais eficiente em redução de potência, devido à maior correlação presente nas entradas dos circuitos multiplicadores.

6 Arquiteturas de Multiplicadores em Complemento de 2

Entre os operadores aritméticos, os módulos multiplicadores são os mais comuns em operações DSP. Neste trabalho, apresentamos uma nova arquitetura para operações de multiplicação em complemento de 2. A arquitetura proposta mantém a forma original de um multiplicador *array* convencional. Todos os bits nos produtos parciais são tratados como bits sem sinal, exatamente como um multiplicador *array* normal, exceto para o último bit de todas as linhas de produto parcial e todos os bits da última linha. Esta arquitetura é estendida para codificação na base 2^m, que permite a redução do número de linhas parciais, proporcionando significativos ganhos em desempenho e redução do consumo de potência. A flexibilidade da nossa arquitetura permite a fácil implementação de multiplicadores para diferentes valores de m. Este mesmo aspecto se torna mais difícil em implementações de multiplicadores Booth, onde implementações para valores de $m > 2$ são complexas. O aspecto da regularidade da nossa arquitetura possibilita a aplicação de outras técnicas de redução de potência. Adicionalmente, neste trabalho é experimentada uma versão *pipelined* das nossas arquiteturas como forma de reduzir a atividade de transição ao longo do *array*. Outro ponto que será abordado neste capítulo é a operação desloca-soma em complemento de 2. Em particular, será apresentada a estrutura da operação de multiplicação desloca-soma em código Híbrido.

6.1 Multiplicadores *Array* em Complemento de 2

Embora as arquiteturas de multiplicadores *array* mostradas no capítulo anterior exibam um bom aspecto de regularidade, estas arquiteturas não levam em consideração operações de multiplicação com sinal. Módulos multiplicadores operando com sinal são comuns em muitas aplicações DSP. Neste contexto, os tipos de multiplicadores mais rápidos são os paralelos. Entre estes, os multiplicadores Wallace [WAL 64] estão entre os mais rápidos. Entretanto, estes multiplicadores não apresentam boa regularidade e não são explorados neste trabalho. Por outro lado, o multiplicador Booth apresenta um bom aspecto de regularidade, além da redução das linhas de produtos parciais. Neste trabalho, as arquiteturas de multiplicador *array* propostas são comparadas com a arquitetura do multiplicador Booth Modificado que representa o estado da arte em operações de multiplicação de baixa potência.

6.1.1 Trabalho Relacionado

Uma grande quantidade de trabalhos de pesquisa visando o projeto de eficientes e regulares arquiteturas de circuitos multiplicadores tem sido desenvolvida, devido a sua complexidade e grande utilização. Esquemas de multiplicação tais como *bi-section*, *Baugh-Wooley* e *Hwang* [HWA 79] propõem a implementação de uma arquitetura em complemento de 2, utilizando módulos repetitivos com padrões de interconexão uniformes. Entretanto, neste tipo de arquitetura não é permitida uma implementação eficiente, devido à forma irregular do tipo árvore-*array* utilizada. O mesmo aspecto da falta de regularidade é observado em [PEK 99], onde é apresentado um esquema de multiplicador baseado em multiplexador. Em [WAN 2001] é

observado um avanço desta técnica, onde a arquitetura exibe um *layout* mais regular do que o apresentado em [PEK 99]. A arquitetura apresenta reduções em área da ordem de 40%. Entretanto, só são observadas reduções em atraso e consumo de potência após a utilização de circuitos somadores mais eficientes.

As técnicas descritas acima têm sido aplicadas à multiplicadores *array* convencionais, cuja operação é realizada bit a bit e algumas vezes a regularidade da arquitetura não é preservada.

Um novo tipo de *array* para um multiplicador paralelo baseado em diferentes agrupamentos de bits de produtos foi proposto em [NAK 86]. Neste trabalho, o esquema proposto necessita de aproximadamente metade das células quando comparado aos multiplicadores *array* anteriores. Apesar do bom desempenho apresentado, a complexidade do circuito aumenta consideravelmente devido aos circuitos contadores utilizados na unidade de adição. Para um multiplicador na base 4, por exemplo, faz-se necessária a utilização de 16 vezes mais *hardware* em relação a um multiplicador *array* binário convencional.

Para aumentar o desempenho dos multiplicadores, considerando aspectos de regularidade e redução do consumo de potência, têm sido propostos projetos de circuitos baseados na técnica de recodificação de Booth [SAM 90, MIL 92, GAL 94, CHE 96, SEI 2001]. No algoritmo Booth Modificado, utiliza-se aproximadamente metade dos produtos parciais.

Apesar do algoritmo de Booth proporcionar uma maior simplicidade para a implementação da sua arquitetura, torna-se difícil projetar arquiteturas para operar em bases maiores do que 4, devido à complexidade em pré-computar, no termo multiplicador, um crescente número de múltiplos do termo multiplicando. Em [SAM 90, MIL 92, GAL 94, SEI 2001] são propostos multiplicadores Booth de alto desempenho baseados em operação em maiores bases. Entretanto, estes circuitos exibem pouca regularidade e não são indicadas reduções no consumo de potência.

Trabalhos de pesquisa direcionados à redução do consumo de potência a partir do multiplicador Booth, operando em bases maiores, são apresentados em [CHE 96, GOL 2000, YU 2000a]. Em [CHE 96], descreve-se uma arquitetura híbrida base-4/base-8 direcionada à multiplicadores de maiores palavras de dados. A arquitetura apresenta 13% menos de consumo de potência para um multiplicador de 64×64 bits. Entretanto, é observado um acréscimo de 9% em valor de atraso. Em [YU 2000a], propõe-se um multiplicador Booth tipo *carry-save array* para baixa potência. Neste trabalho, mostra-se que é posível obter uma redução de 18% em potência. Entretanto, não é apresentado nenhum acréscimo em desempenho. Em [GOL 2000], apresenta-se um multiplicador que utiliza codificadores Booth na base 4, que são otimizados para geração dos produtos parciais. São apresentados resultados com reduções em área, atraso e consumo de potência com um esquema de codificação altamente otimizado no nível de transistores. Neste nível de abstração, alguns outros trabalhos têm abordado multiplicadores direcionados à reduções em área [GOT 97], desempenho [YAN 90, KHA 96] e potência [KHA 96]. Em todos estes trabalhos citados, aplica-se lógica de transistores de passagem complementar aos seus projetos, para melhorar o esquema de codificação Booth e os circuitos somadores completos.

No nosso trabalho, buscam-se os mesmos objetivos da arquitetura Booth, ou seja, alcançar melhores desempenhos e menores consumos de potência a partir da redução dos termos dos produtos parciais, mantendo-se a mesma regularidade de

um multiplicador *array* [COS 2002, COS 2002a]. Nós mostramos que nossas arquiteturas podem ser mais naturalmente estendidas à operações de multiplicação em bases maiores utilizando menos níveis lógicos e, desta forma, apresentando menos transições espúrias. No nosso trabalho, não se aplica ainda nenhuma técnica no nível de transistores, o que pode aumentar mais a eficiência das arquiteturas propostas.

6.1.2 Multiplicador na Base 2^m

Nesta parte do trabalho serão apresentadas as arquiteturas *array* em complemento de 2 operando nos códigos Binário e Híbrido. Como será mostrado, o mesmo aspecto de regularidade das arquiteturas *array* para operações sem sinal, mostrado no capítulo anterior, pode ser aplicado às arquiteturas *array* em complemento de 2.

6.1.2.1 Multiplicação Binária em Complemento de 2

Em aplicações de processamento de sinais, a representação de complemento de 2 é uma das codificações mais utilizada para operações de números com sinal. Para este tipo de operação, o bit mais significativo a_{W-1} representa o bit de sinal. Neste caso, se o número A é positivo, sua representação é a mesma da utilizada para um número sem sinal, ou seja, simplesmente A. Entretanto, se o número é negativo, sua representação é dada por $2^W - A$. Deste modo, o valor do operando pode ser computado da seguinte forma:

$$\begin{cases} A & , \quad a_{W-1} = 0 \\ A - 2^W & , \quad a_{W-1} = 1 \end{cases} \tag{6.1}$$

Definindo $A' = \sum_{i=0}^{W-2} a_i 2^i$, como um valor sem sinal. Para números positivos, $a_{W-1} = 0$, e desta forma o valor representado por A é A'. Para números negativos, $a_{W-1} = 1$, e desta forma este valor é $A - 2^W = 2^{W-1} + A' - 2^W = A' - 2^{W-1}$. Logo, a Equação 6.1 se torna:

$$\begin{cases} A' & , \quad a_{W-1} = 0 \\ A' - 2^{W-1} & , \quad a_{W-1} = 1 \end{cases} \tag{6.2}$$

ou simplesmente, $A' - a_{W-1}2^{W-1}$.

De fato, o que a Equação 6.2 mostra é que a multiplicação de dois operandos em complemento de 2 pode ser realizada como uma multiplicação sem sinal para $(W-1)^2$ dos produtos dos bits. As 4 possibilidades para $A \times B$ são dadas por:

$$\begin{array}{ll} A > 0, B > 0: & A' \times B' \\ A > 0, B < 0: & A' \times B' - A'2^{W-1} \\ A < 0, B > 0: & A' \times B' - \sum_{j=0}^{W-1} b_j 2^{W-1+j} \\ A < 0, B < 0: & A' \times B' - A'2^{W-1} - \sum_{j=0}^{W-1} b_j 2^{W-1+j} \end{array} \tag{6.3}$$

Considerando as possibilidades acima, pode-se ter uma expressão reduzida, que é dada por:

$$A \times B = A' \times B' - b_{W-1}A'2^{W-1} - a_{W-1} \sum_{j=0}^{W-1} b_j 2^{W-1+j} \tag{6.4}$$

A forma da Equação 6.4 destaca:

- para o primeiro termo, tem-se que os $W-1$ bits menos significativos de A e B podem ser tratados exatamente como um multiplicador *array* sem sinal;
- para o segundo termo, tem-se que a última linha do multiplicador pode não existir (para o caso de $B > 0$) ou ser um subtrator de A' deslocado por $W-1$ bits (para o caso de $B < 0$);
- para o terceiro termo, tem-se que em cada linha de produto parcial, o bit mais significativo pode ser 0 ($A > 0$) ou -1 ($A < 0$).

Na Figura 6.1, apresenta-se a operação de uma multiplicação *array* de operandos de 4 bits em complemento de 2, considerando os aspectos descritos acima.

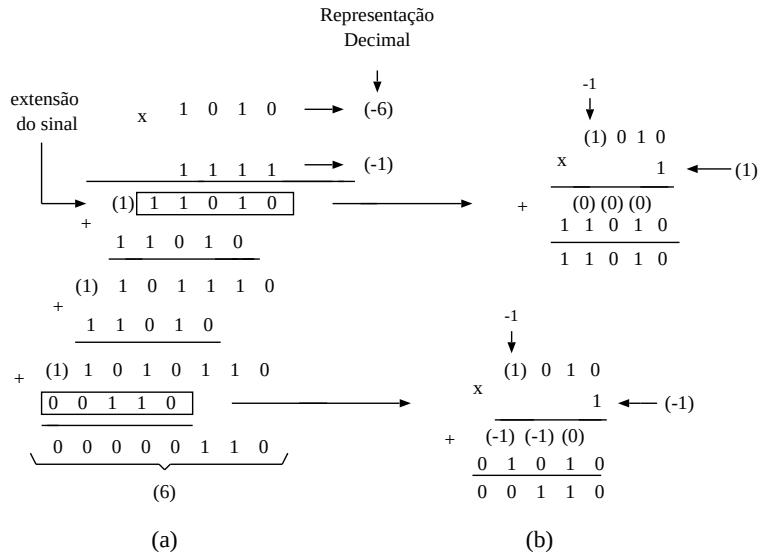

FIGURA 6.1 – Exemplo de uma multiplicação Binária com sinal em operandos $W = 4$ bits.

A partir do exemplo de operação Binária em complemento de 2 mostrado na Figura 6.1, pode-se construir um multiplicador tipo *array* que produza operandos com sinal. Isto é realizado simplesmente usando poucos elementos diferentes no lado esquerdo e na parte inferior do *array*, como mostram as linhas pontilhadas da Figura 6.2. Nesta figura, apresenta-se uma arquitetura para operandos de 4 bits. Deve ser observado na Figura 6.2, que para o caso de W=4, tem-se um total de 16 bits de produto, dos quais 9 são sem sinal e 7 com sinal. Entretanto, esta taxa, $(W-1)^2$ vs. $2W-1$, aumenta com W. No caso de um multiplicador de 16 bits, por exemplo, tem-se 225 bits de produto sem sinal e 31 bits com sinal.

6.1.2.2 Multiplicação Binária na Base 2^m em Complemento de 2

Neste trabalho, pode-se demonstrar que todas as observações feitas na seção anterior são aplicáveis a qualquer base e qualquer representação de dados. Considerando agora $A' = \sum_{i=0}^{\frac{W}{m}-2} a_i 2^{i \cdot m}$, onde a_i é um dígito de m-bits. Para números positivos, o valor representado por A é A' como antes. Para números negativos, este valor é $A - 2^W = a_{\frac{W}{m}-1} 2^{W-m} + A' - 2^W = A' - a_{\frac{W}{m}-1} 2^{W-m}$, visto que $a_{\frac{W}{m}-1} 2^{W-m} - 2^W$ é

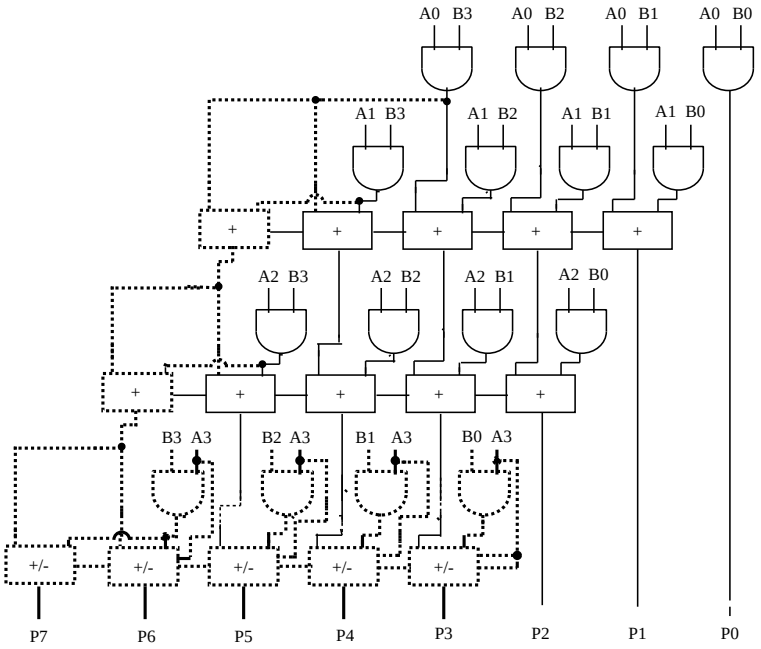

FIGURA 6.2 – Exemplo de um multiplicador *array* Binário com sinal para operandos $W = 4$ bits.

o complemento de 2 de $a_{\frac{W}{m}-1}2^{W-m}$. Logo, tem-se que o valor do operando é dado por:

$$\begin{cases} A' & , \quad a_{W-1} = 0 \\ A' - a_{\frac{W}{m}-1}2^{W-m} & , \quad a_{W-1} = 1 \end{cases} \tag{6.5}$$

ou simplesmente,

$$A' - a_{W-1}a_{\frac{W}{m}-1}2^{W-m} \tag{6.6}$$

Usando observações análogas às feitas para a representação da Equação 6.6, pode-se escrever:

$$\begin{aligned} A \times B &= A' \times B' - A'b_{W-1}b_{\frac{W}{m}-1}2^{W-m} \\ &\quad -a_{W-1}a_{\frac{W}{m}-1}\sum_{j=0}^{\frac{W}{m}-1} b_j 2^{W-m+j} \end{aligned} \tag{6.7}$$

Um exemplo desta operação é mostrado na Figura 6.3, para uma multiplicação Binária de 8 bits na base 16.

Como pode ser visto na Figura 6.3, mais uma vez para os $W - m$ bits menos significativos dos operandos a multiplicação sem sinal pode ser utilizada. De acordo com os bits mais significativos do operando, pode-se ter operações envolvendo números com sinal. A utilização da técnica de extensão do sinal [MOS 95, ANG 96] permite manter a regularidade da estrutura. No exemplo mostrado na Figura 6.3, deve ser observado que há três tipos de operação envolvidos, ou seja, (1) operação do termo multiplicador positivo com o multiplicando positivo, (2) operação do termo

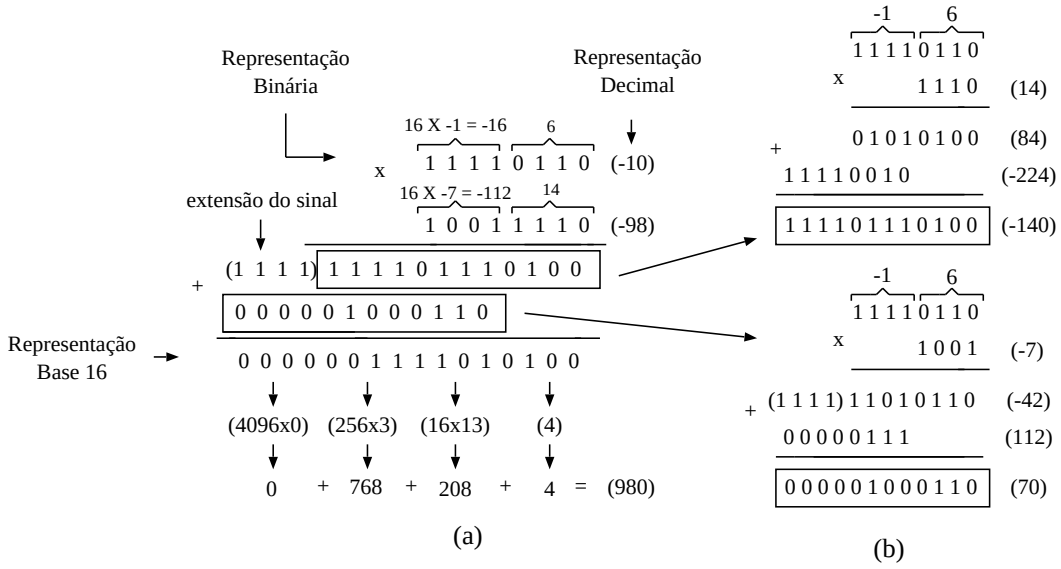

FIGURA 6.3 – Exemplo de multiplicação Binária de 8 bits base 16 em complemento de 2.

multiplicador positivo com o multiplicando negativo e (3) operação do termo multiplicador negativo com o multiplicando negativo. Estas operações podem ser vistas na Figura 6.3(b).

6.1.2.3 Multiplicação Híbrida na Base 2^m em Complemento de 2

A idéia de dividir os operandos em grupos de m bits e codificar cada grupo utilizando o código Gray pode ser também utilizada para operandos que operam na representação em complemento de 2. A Tabela 6.1 mostra as codificações Binária, Híbrida (m=2) e Gray em complemento de 2 para números de 4 bits.

A multiplicação Híbrida em complemento de 2 é realizada da mesma forma mostrada na Figura 6.3 para a operação Binária. A diferença consiste na aplicação do código Gray em complemento de 2 aos grupos de m bits positivos e negativos. Outra diferença a ser destacada é a aplicação do código Gray à extensão do sinal, como pode ser visto no exemplo da Figura 6.4(a). Nesta figura pode ser observado que o valor da extensão do sinal corresponde ao valor -1 em código Gray m=4 (1000), em oposição ao valor 1111 do Binário, como mostrado na Tabela 6.1.

6.1.2.4 Arquitetura de Multiplicador na Base 2^m em Complemento de 2

Para a construção de uma arquitetura *array* na base 2^m, os módulos do produto parcial no lado esquerdo e na parte inferior precisam ser diferentes para produzir o sinal dos operandos. De forma a acomodar estas situações são construídos três tipos de módulos. `Tipo I` são os módulos sem sinal usados no Capítulo 5. Os módulos `Tipo II` produzem os produtos parciais de m-bits de uma operação de um valor sem sinal com um valor em complemento de 2. Finalmente, os módulos `Tipo III`, que operam em dois valores com sinal. Deve-se destacar que, para multiplicadores na base 2^m, são necessários $2\frac{W}{m} - 2$ módulos `Tipo II` e $(\frac{W}{m} - 1)^2$ módulos `Tipo I`. Por

TABELA 6.1 – Representações dos códigos Binário, Híbrido (m=2) e Gray em complemento de 2 para números de 4 bits.

Decimal	Binário	Híbrido (m=2)	Gray
0	0000	0000	0000
1	0001	0001	0001
2	0010	0011	0011
3	0011	0010	0010
4	0100	0100	0110
5	0101	0101	0111
6	0110	0111	0101
7	0111	0110	0100
-8	1000	1100	1100
-7	1001	1101	1101
-6	1010	1111	1111
-5	1011	1110	1110
-4	1100	1000	1010
-3	1101	1001	1011
-2	1110	1011	1001
-1	1111	1010	1000

outro lado, para qualquer tamanho de circuito multiplicador (Binário ou Híbrido), é necessário apenas um módulo `Tipo III`.

A arquitetura geral para um multiplicador base 2^m em complemento de 2 é mostrado na Figura 6.5. Na Figura 6.6 é mostrado um exemplo concreto para operandos de largura $W = 8$ na representação usando base 16 ($m = 4$).

Como pode ser visto no exemplo da Figura 6.6, a arquitetura de um multiplicador *array* na base 2^m, em complemento de 2, exibe a mesma estrutura de um circuito sem sinal, mostrado na Figura 5.13. A diferença consiste na aplicação dos diferentes módulos, que compõem a arquitetura em complemento de 2. Entretanto, cabe ressaltar que, embora na arquitetura em complemento de 2 sejam utilizados 3 módulos diferentes (Tipo I, Tipo II e Tipo III), estes apresentam praticamente a mesma complexidade, tendo-se desta forma resultados muito próximos das arquiteturas sem sinal, como será mostrado posteriormente. Além dos diferentes módulos utilizados, a arquitetura em complemento de 2 inclui as ligações para estabelecimento da técnica de extensão do sinal, como mostrado na linha pontilhada da Figura 6.6.

Para uma arquitetura em código Híbrido, a estratégia a ser adotada é a utilização do código Gray em complemento de 2 nos grupos de m bits. Além disso, a arquitetura deve incluir a técnica de extensão do sinal em código Gray, como mostrado na linha pontilhada da Figura 6.7. Como pode ser visto nesta figura, para uma arquitetura em código Híbrido, com m=4, os três bits menos significativos do último elemento somador da última linha de soma dos termos do produto parcial permanecem em nível lógico 0. O último bit da operação de soma anterior é ligado ao bit mais significativo deste último elemento somador, definindo assim o sinal da próxima soma. Para os demais valores de m, adota-se esta mesma estratégia. Isto de deve ao fato de que independente do valor de m, para um valor decimal igual a -1, o código Gray sempre apresenta apenas o bit mais significativo em nível lógico 1 e o restante em nível lógico 0.

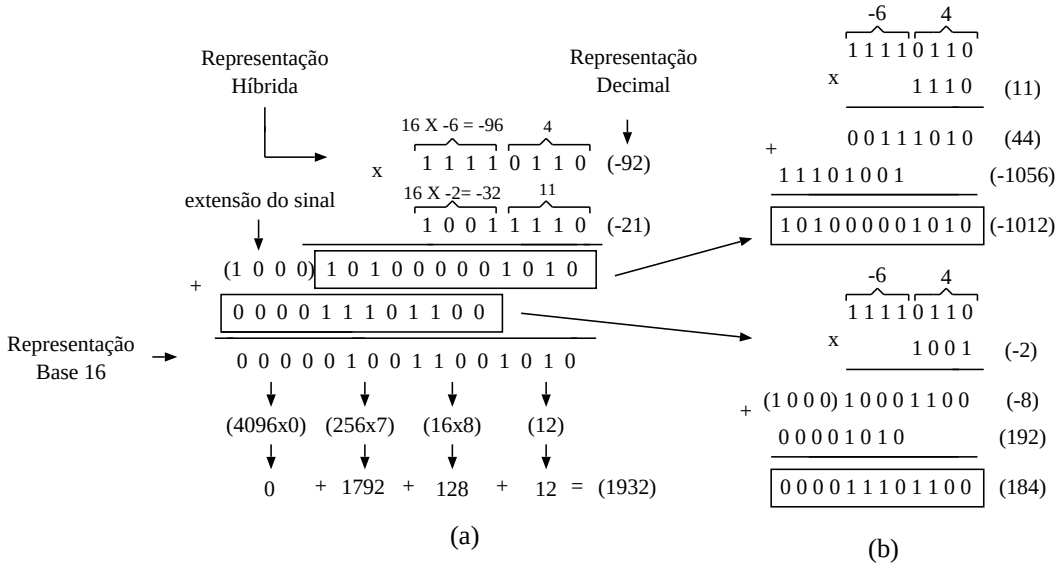

FIGURA 6.4 – Exemplo de multiplicação Híbrida de 8 bits base 16 em complemento de 2.

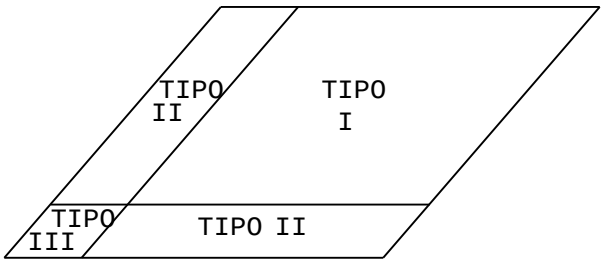

FIGURA 6.5 – Estrutura geral de um multiplicador base 2^m em complemento de 2.

6.1.3 Multiplicador Booth Modificado

Nesta parte do trabalho serão abordados os aspectos de multiplicação do algoritmo Booth. Por representar o estado da arte em operações de baixa potência, nos concentraremos neste multiplicador para comparações com as arquiteturas de multiplicador *array* propostas neste trabalho.

O algoritmo Booth verifica a deteção do início, do interior e do final de uma cadeia de valores no nível lógico 1. Este algoritmo foi proposto em [BOO 51], e apresenta como principal característica a possibilidade de operar sobre números representados em complemento de 2. No algoritmo Booth original, 2 bits são verificados ao mesmo tempo, determinando o tipo de operação a ser efetuada. Desta forma, em um termo multiplicador, o conjunto de dígitos que compõe a representação na base 2 é dado por (-1,0,1). Este conjunto é utilizado para ajustar os termos multiplicando. Uma codificação do termo multiplicando é utilizada para simplificar os produtos parciais.

O algoritmo Booth Modificado alcança um melhor desempenho através da codificação na base 4. Neste algoritmo são gerados os sinais de controle (0,+X,+2X,-X e -2X) a partir do operando do multiplicador para cada grupo de 3 bits, como mostra o exemplo da Figura 6.8 para uma operação de 8 bits. Um circuito multiplexador

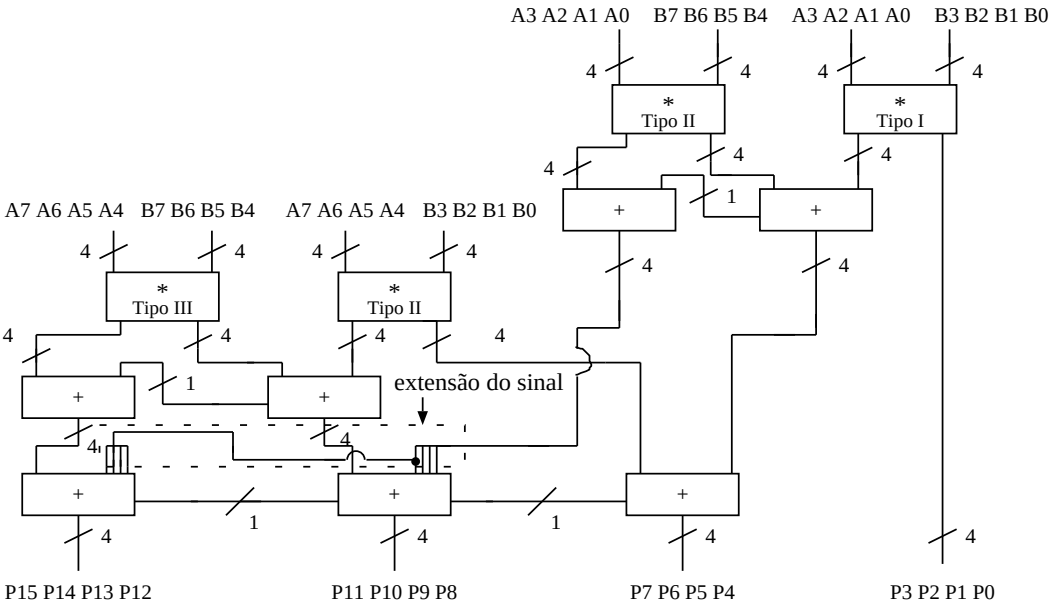

FIGURA 6.6 – Exemplo de multiplicador *array* Binário de 8 bits na base 16 em complemento de 2.

produz os produtos parciais, de acordo com o sinal de controle codificado.

Como pode ser observado no exemplo da Figura 6.8, para reduzir o número de produtos parciais no multiplicador Booth Modificado, realiza-se uma operação em grupos de 3 bits no termo multiplicador (MR). De fato, o termo multiplicador é verificado de 2 em 2 bits com sobreposição do terceiro bit. Na varredura dos bits menos significativos do termo multiplicador, considera-se a existência de um valor zero virtual à direita do bit menos significativo. Deve-se observar que no algoritmo Booth também é utilizada a técnica de extensão do sinal. Para este algoritmo, utilizam-se dois bits extras nos produtos parciais, para tornar o *array* mais regular. As regras que habilitam as operações sobre o produto parcial (PP), de acordo com os bits do multiplicador, são mostradas na Tabela 6.2 [HWA 79, WU 98a]. Como pode ser observado nesta tabela, para os números decimais negativos -1 e -2 é necessário gerar o complemento de 2 para o termo multiplicando. Isto é realizado pela negação do multiplicando e adição do valor 1 a este termo.

TABELA 6.2 – Regras de operação do multiplicador Booth Modificado.

Multiplicador	Decimal	Operação no Multiplicando
000	0	adição de 0 ao produto parcial
001, 010	1	adição do multiplicando ao produto parcial
011	2	deslocamento do multiplicando à esquerda e adição ao produto parcial
100	-2	complemento de 2 do multiplicando, deslocamento à esquerda e adição ao produto parcial
101, 110	-1	complemento de 2 do multiplicando e adição ao produto parcial

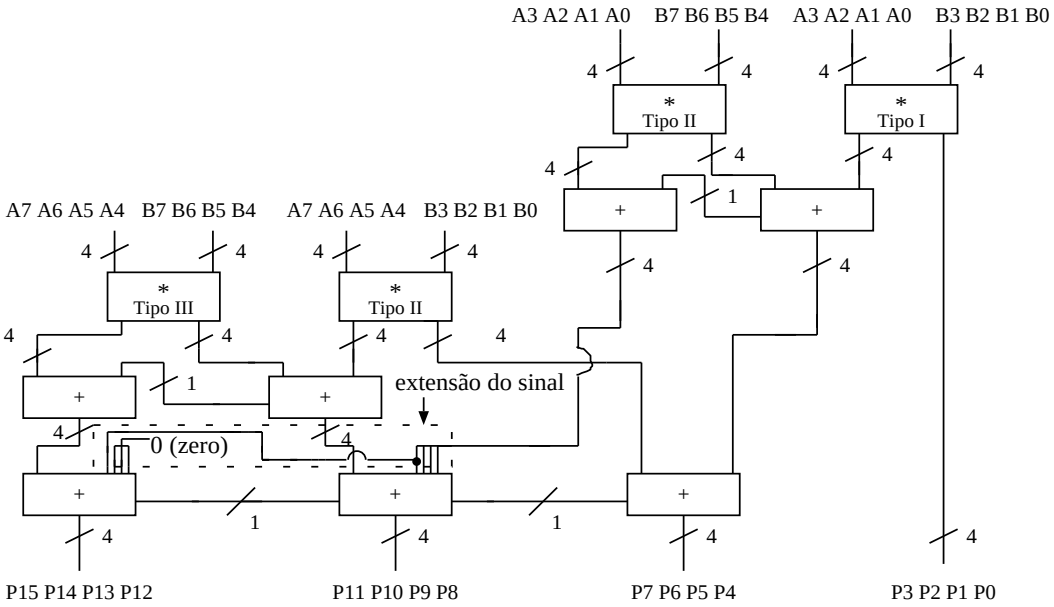

FIGURA 6.7 – Exemplo de multiplicador *array* Híbrido de 8 bits na base 16 em complemento de 2.

A arquitetura do algoritmo Booth na base 4 (Booth Modificado) é apresentada em [KHA 96]. Nesta arquitetura é possível reduzir o número de produtos parciais, pela codificação do multiplicador em complemento de 2. Um dos principais problemas no multiplicador Booth Modificado é a geração do complemento de 2 para o termo multiplicando. Isto é calculado por cada circuito operando, mostrado na Figura 6.9, para uma operação de 8 bits. Como pode ser observado nesta figura, para uma arquitetura de 8 bits são necessários 4 circuitos operandos para calcular os termos dos produtos parciais. Esses circuitos são compostos por um codificador e um multiplexador, que produzem o termo multiplicando de acordo com os 3 bits no termo multiplicador (MR). Os termos do produto parcial são deslocados pelos circuitos somadores, que são também utilizados para produzir o resultado final.

O aspecto comum entre a arquitetura Booth e as arquiteturas propostas neste trabalho é o fato de que em cada passo do algoritmo dois bits são processados ao mesmo tempo. Entretanto, as células Booth básicas não são simples circuitos somadores como nas arquiteturas *array* propostas neste trabalho. Estas células têm de realizar operações de soma, subtração e deslocamento à esquerda, dos bits do termo multiplicando. Esta complexidade torna difícil a implementação de arquitetura Booth em bases maiores do que 4.

6.1.4 Comparações de Desempenho

Como mostrado na seção anterior, o uso de diferentes grupos de m bits permite a implementação de arquiteturas de multiplicadores regulares em códigos Binário e Híbrido, que operam em complemento de 2. Nesta seção são comparados inicialmente valores de área, atraso e potência para multiplicadores *array* de 16 bits, para grupos de bits m=1,2 e 4. Posteriormente, são apresentados resultados de comparação da arquitetura Booth base 4 com as arquiteturas *array* usando (m=2). Os resultados para operadores com o grupo m=8 não são apresentados nesta seção,

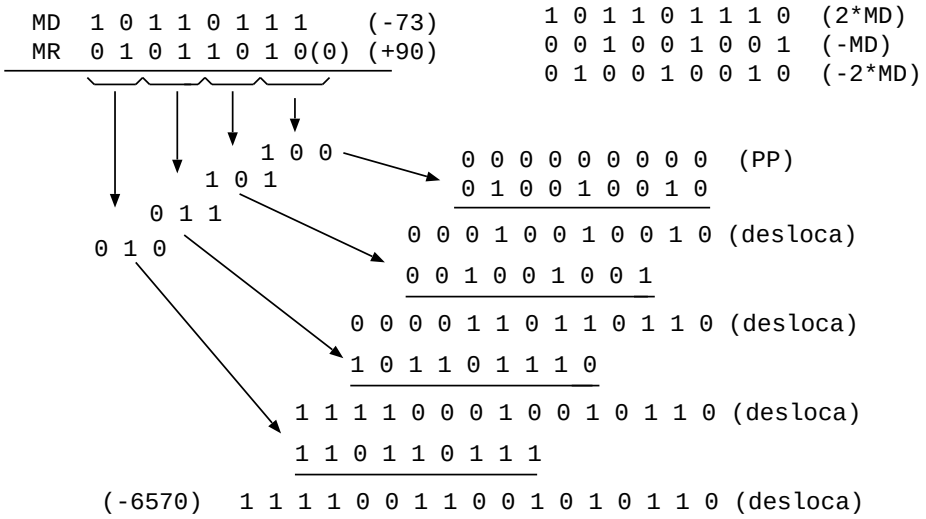

FIGURA 6.8 – Exemplo de multiplicação de 8 bits usando o algoritmo Booth Modificado.

devido a grande complexidade destes operadores, como mostrado na Seção 5.1.5.4.

6.1.4.1 Multiplicadores *Array* com Sinal Usando Diferentes Bases

Como visto anteriormente, torna-se possível gerar arquiteturas de circuitos que operam em complemento de 2 com uma estrutura muito semelhante às arquiteturas sem sinal. Desta forma, os resultados obtidos para as arquiteturas de multiplicadores *array* convencional m=1 e multiplicadores *array* Binário e Híbrido na base 2^m, em complemento de 2, são muito próximos aos resultados apresentados para estas mesmas arquiteturas sem sinal. Embora a complexidade dos módulos utilizados nas arquiteturas em complemento de 2 sejam praticamente as mesmas, a necessidade da utilização de circuitos somadores/subtratores nos módulos dos termos dos produtos faz com que estas arquiteturas apresentem maiores valores de parâmetros de área, atraso e potência, comparado às arquiteturas sem sinal. Entretanto, são mantidas as mesmas diferenças de valores entre os grupos de bits, ou seja, as arquiteturas com valores de m=2 e 4 apresentam maiores valores de área em relação à arquitetura com m=1, como mostrado na Tabela 6.3. Isto devido à maior complexidade dos módulos básicos com valores de m maiores do que 1.

TABELA 6.3 – Área para multiplicadores Binário e Híbrido em complemento de 2.

Grupo de bits	Binário		Híbrido	
	Literais	%	Literais	%
m=1	4266	–	–	–
m=2	4674	+9,5	5316	+24,6
m=4	10544	+147,1	22173	+419,7

O cálculo do número de elementos presentes no caminho crítico de uma arquitetura na base 2^m, apresentado na Seção 5.1.5.3, também é válido para as arquiteturas em complemento de 2. A diferença consiste na última linha das arquiteturas,

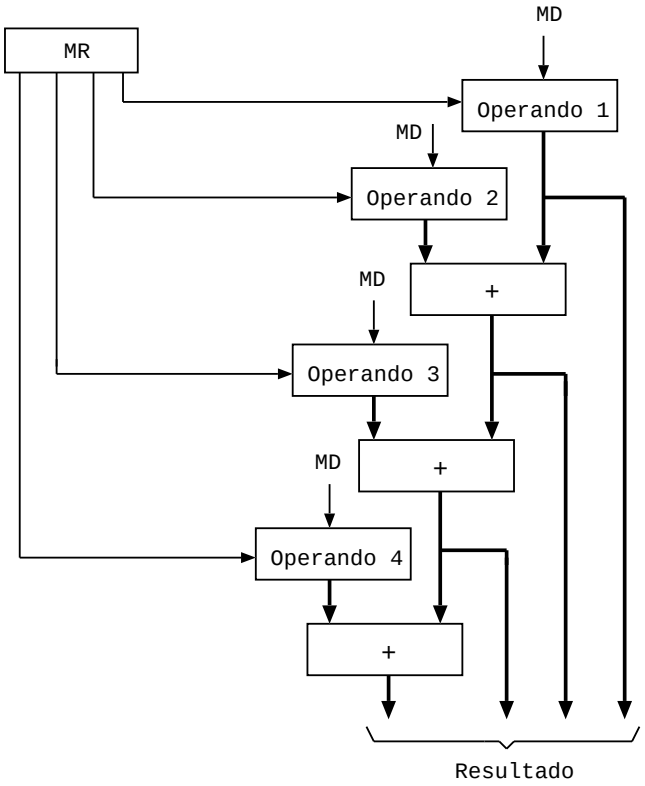

FIGURA 6.9 – Exemplo de arquitetura Booth Modificada de 8 bits.

que soma ou subtrai os termos do produto, dependendo do último bit do termo multiplicador. Estes circuitos (somadores/subtratores) presentes no caminho crítico elevam os valores de atraso das arquiteturas em complemento de 2 em relação às arquitetura sem sinal. Entretanto, a relação dos valores de atraso entre os grupos de bits é mantida, ou seja, é possível reduzir os atrasos das arquiteturas com m=2 e m=4, como mostrado na Tabela 6.4. Neste particular, a arquitetura Binária, com m=4, apresenta o menor valor de atraso, devido à menor quantidade de elementos presentes no caminho crítico.

TABELA 6.4 – Atraso para multiplicadores Binário e Híbrido em complemento de 2.

Grupo de bits	Binário		Híbrido	
	Atraso(ns)	%	Atraso(ns)	%
m=1	284,0	–	–	–
m=2	231,1	-18,6	231,6	-18,4
m=4	201,7	-28,9	246,1	-13,3

Uma das principais características apresentada pelas arquiteturas na base 2^m sem sinal é a possibilidade de redução das linhas de produtos parciais, que reduz o caminho crítico e a quantidade de transições espúrias ao longo do *array*. Como pôde ser visto nos resultados apresentados para estas arquiteturas, este aspecto é fundamental para a redução de potência dos circuitos. Para as arquiteturas na base 2^m em complemento de 2, esta característica de redução das linhas de produto parcial

se mantém válida, devido à técnica de extensão do sinal que mantém a regularidade do *array*.

De fato, o aspecto da redução de linhas dos produtos parciais contribui para a redução da profundidade lógica dos circuitos na base 2^m, também em complemento de 2, devido aos caminhos mais balanceados para os blocos básicos que compõem as arquiteturas com m>1. Este aspecto é fundamental para a redução de potência destas arquiteturas, como mostrado na tabela 6.5. Para m=4, a arquitetura *array* Binária produz significativamente maior redução de potência, devido ao menor atraso apresentado por esta arquitetura, que produz a geração de menos transições espúrias ao longo dos circuitos.

TABELA 6.5 – Potência para multiplicadores Binário e Híbrido em complemento de 2 com entradas de *trace* real.

Grupo de bits	Binário		Híbrido	
	Potência(mW)	%	Potência(mW)	%
m=1	134,23	–	–	–
m=2	56,52	-57,8	55,78	-58,4
m=4	36,43	-72.8	56,06	-58,2

Como pode ser visto na Tabela 6.6, mesmo com a aplicação de sinais com um padrão de entradas randômicas, onde o aspecto da correlação não está presente, é possível reduzir potência das arquiteturas *array* com valores de m=2 e 4. Entretanto, deve ser observado que o aspecto da falta de correlação dos sinais reduz os valores de ganho em potência em relação às entradas senoidais.

TABELA 6.6 – Potência para multiplicadores Binário e Híbrido em complemento de 2 com entradas randômicas.

Grupo de bits	Binário		Híbrido	
	Potência(mW)	%	Potência(mW)	%
m=1	201,0	–	–	–
m=2	89,0	-55,7	94,1	-53,1
m=4	76,4	-61,9	134,8	-32,8

6.1.4.2 Multiplicadores *Array* em Complemento de 2 na Base 4

No multiplicador Booth, usa-se um esquema de codificação para reduzir o número de estágios na multiplicação. Neste esquema, dois bits da multiplicação são processados ao mesmo tempo, e desta forma o multiplicador necessita de metade dos estágios. Nos multiplicadores *array* propostos neste trabalho, o número de estágios pode ser reduzido para além da metade, enquanto que a regularidade pode ser mantida como em um circuito multiplicador *array* normal. A Tabela 6.7 apresenta resultados de área, atraso, níveis lógicos e potência para os multiplicadores Booth base 4 e os multiplicadores Binário e Híbrido com m=2.

Como pode ser observado na Tabela 6.7, os multiplicadores *array* m=2 apresentam os maiores valores de área. Isto ocorre devido ao fato das linhas de produto parcial operarem em grupos de m bits e os elementos multiplicadores básicos, que

TABELA 6.7 – Área, níveis lógicos, atraso e potência para multiplicadores paralelos de 16 bits em complemento de 2.

	Área	%	Atraso (ns)	%	Níveis Lógicos	%	Potência (mW)	%
Binário	4674	–	231,1	–	74	–	56,5	–
Híbrido	5316	+19,4	231,6	+0,2	76	+2,7	55,7	-1,3
Booth	3912	-20,2	233,7	+4,7	81	+9,4	76,7	+35,7

compõem os módulos dos termos dos produtos, serem mais complexos. Neste particular, o multiplicador *array* Híbrido apresenta o maior valor de área, devido à maior complexidade dos elementos básicos.

Apesar dos multiplicadores *array* apresentarem maiores valores de área, estas arquiteturas podem apresentar menores valores de atraso, como mostrado na Tabela 6.7. Isto devido à menor quantidade de termos de produtos parciais e menor valor de níveis lógicos apresentados por estes circuitos, como mostrado na Tabela 6.7. Estes fatores estão associados, principalmente, ao fato do multiplicador Booth necessitar de um esquema de codificação e um multiplexador, que produzem o termo multiplicando de acordo com os 3 bits no termo multiplicador. Estes circuitos no multiplicador Booth apresentam uma maior complexidade, com uma maior quantidade de interconexões e maior valor de atraso comparado ao módulo multiplicador básico das arquiteturas *array*.

Como observado em [CAL 93], a maior fonte de dissipação de potência em circuitos multiplicadores são transições espúrias e corridas lógicas que fluem através do *array*. Desta forma, o maior número de interconexões presente no multiplicador Booth é responsável pela geração de uma significativa quantidade de *glitching* no circuito, que justifica um grande ganho em potência dos multiplicadores *array* Binário e Híbrido, como observado na Tabela 6.7. Para confirmar este aspecto foram realizadas estimativas de potência com as arquiteturas *array* e Booth, usando modelo de atraso zero, e os valores obtidos foram praticamente os mesmos. Outro aspecto a ser destacado na Tabela 6.7 é que o multiplicador *array* Híbrido apresenta menor valor de consumo de potência em relação ao multiplicador Binário, mesmo apresentando maiores valores de área, atraso e níveis lógicos. Isto ocorre devido ao tipo de sinal de *trace* real aplicado, que apresenta uma maior correlação. Esta maior correlação faz com que o sinal em código Híbrido apresente uma menor quantidade de transições, comparado ao mesmo tipo de sinal em código Binário. Este mesmo aspecto não ocorre quando aplica-se às entradas dos circuitos um sinal de padrão randômico. Neste caso, o aspecto da falta de correlação no sinal randômico faz com que a arquitetura *array* Híbrida apresente maior valor de consumo de potência em relação à arquitetura Binária, como mostrado na Tabela 6.8.

Outro aspecto a ser visto na Tabela 6.8 é o aumento do consumo de potência da arquitetura Booth em relação à arquitetura *array* Binária, acima do percentual anteriormente mostrado para entradas de *trace* real (Tabela 6.7). Isto é devido à possibilidade de aumento do efeito de propagação de *glitching* com a maior variação dos bits no sinal randômico.

TABELA 6.8 – Potência para multiplicadores paralelos em complemento de 2 com sinais randômicos.

Arquitetura	Potência(mW)	%
Array Binário	89,0	–
Array Híbrido	94,1	+5,7
Modified Booth	137,0	+53,9

6.1.5 Arquiteturas na Forma *Pipelined*

Os circuitos multiplicadores na forma *pipelined* são úteis no caso do *throughput* aritmético ser mais importante do que a latência. Isto devido ao fato da introdução de registradores ao longo do *array* reduzir a quantidade de atividade de chaveamento desnecessária. Um multiplicador *pipelined* pode realizar mais operações por segundo e operar com uma taxa de relógio mais elevada do que um multiplicador não *pipelined*, logo este tipo de multiplicador (*pipelined*) é desejável para aplicações aritméticas de alto desempenho [ASA 90].

6.1.5.1 Trabalho Relacionado

Alguns trabalhos têm abordado o aspecto de inserção de registradores nos circuitos multiplicadores no sentido de reduzir a quantidade de transições espúrias ao longo do *array*. Em [MUS 95], apresenta-se um método para redução de potência em multiplicadores *array*, baseado na inserção de *transition-retaining barriers* (TRB). De acordo com os autores, com a utilização de três TRBs em um multiplicador *array* de 32×32 bits, é possível obter uma redução de dissipação de potência na ordem de 30%. Entretanto, a inserção destes elementos ao longo do circuito produz aumentos nos parâmetros de área e atraso. Em [LEM 94], utilizam-se *latches* em um multiplicador Booth base 4 para redução da atividade de chaveamento. Nesta técnica, cada *latch* é habilitado utilizando uma linha de atraso. O sucesso da técnica depende da forma como a linha de atraso trata o atraso de propagação das somas e os bits de *carry* ao longo do *array*. De acordo com os autores, torna-se possível uma redução de 40% em consumo de potência com a arquitetura operando em uma tensão de alimentação de 3V. Em [LEI 95], analisa-se a influência da atividade de transição em multiplicadores *array*, devido à lógica combinacional, *flip-flops* e linhas de relógio e a partir das análises considerando estes elementos, sugere-se um nível ótimo de *pipelining*.

No nosso trabalho, são implementadas arquiteturas na forma *pipelined* para verificar o relacionamento entre o maior desempenho e a redução do consumo de potência [COS 2002b]. Como será mostrado, apesar da maior área exibida pelas nossas arquiteturas na forma *pipelined*, devido à introdução de registradores ao longo do *array*, é possível se ter arquiteturas operando com melhor desempenho e menor consumo de potência em relação às arquiteturas não *pipelined*. As nossas arquiteturas são comparadas com uma versão *pipelined* da arquitetura Booth Modificada.

6.1.5.2 Arquitetura *Array* Base 2^m na Forma *Pipelined*

A regularidade das arquiteturas propostas neste trabalho permite a aplicação de outras técnicas de redução de potência. Neste trabalho, apresenta-se a implementação de uma versão *pipelined* das nossas arquiteturas, como forma da redução do caminho crítico e redução de transições espúrias que são propagadas através do *array*. As linhas pontilhadas na Figura 6.10 mostram a versão *pipelined* do multiplicador *array* na base 4 para operandos de 8 bits. Esta arquitetura pode ser aplicada à operações em códigos Binário ou Híbrido.

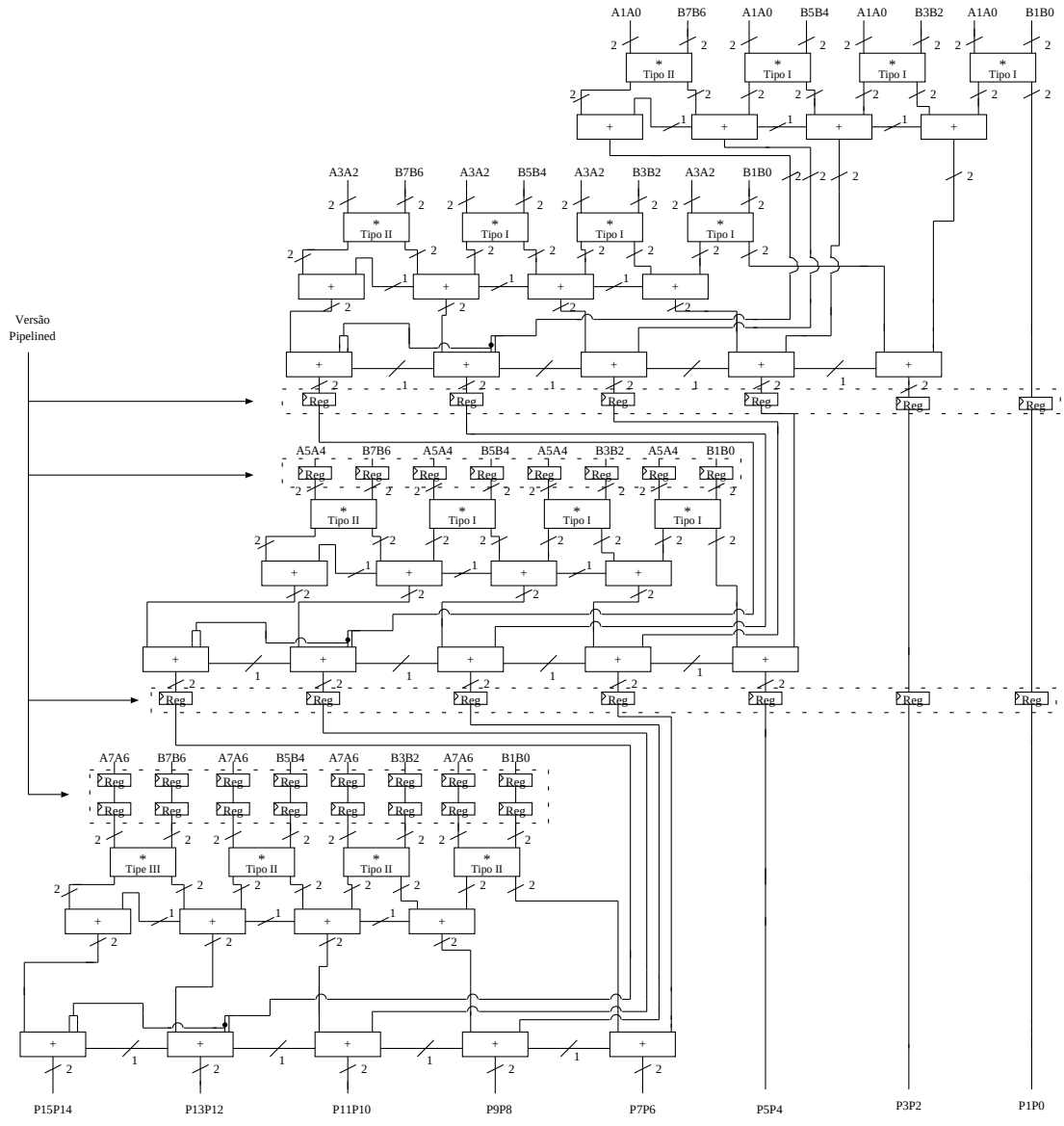

FIGURA 6.10 – Exemplo de um multiplicador *array* de 8 bits base 4 na forma *pipelined*.

Como pode ser observado na Figura 6.10, o aspecto da regularidade da arquitetura permite a introdução de duas camadas de registradores distribuídas ao longo do *array*. Desta forma, para este exemplo, as operações são efetuadas em 3 ciclos de relógio. Para os multiplicadores *array* na base 2^m, com W bits, são utilizadas $\frac{W}{m} - 2$ camadas de registradores. Desta forma, as operações são realizadas em $\frac{W}{m} - 1$ ciclos

de relógio. Como será apresentado posteriormente, a introdução de registradores ao longo do circuito aumenta a área significativamente, tendo-se entretanto reduções em valores de atraso e consumo de potência.

Deve-se observar que os resultados que serão apresentados com esta implementação *pipelined*, podem ser extrapolados para uma implementação serial, onde quase não há efeito de propagação de *glitching*.

6.1.5.3 Arquitetura Booth Modificada na Forma *Pipelined*

Neste trabalho, apresenta-se também uma versão *pipelined* da arquitetura Booth Modificada, como mostra a Figura 6.11. Esta figura utiliza o mesmo exemplo anterior, ou seja, operação em 8 bits na base 4.

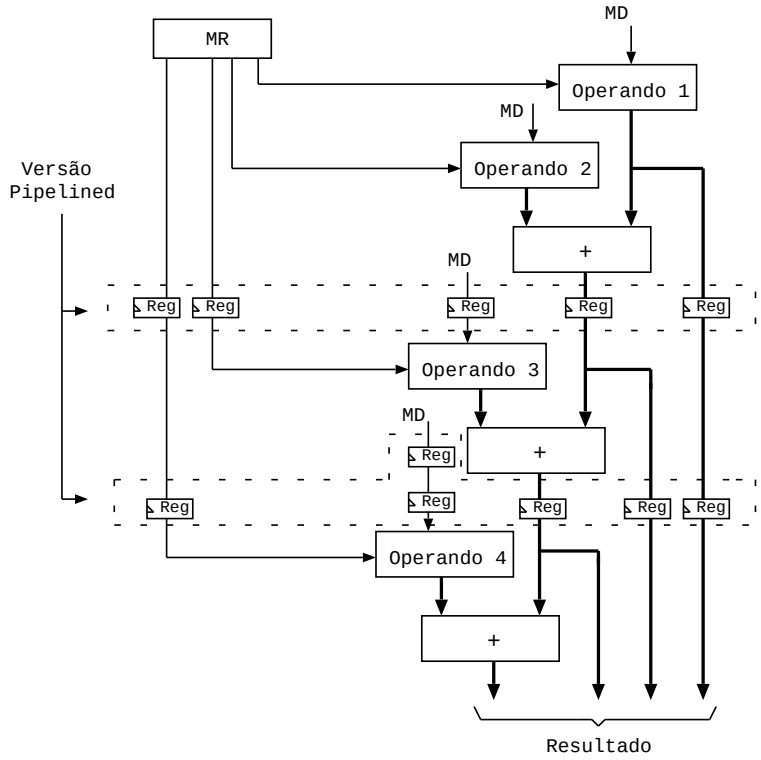

FIGURA 6.11 – Exemplo de arquitetura Booth Modificada de 8 bits na forma *pipelined*.

Como pode ser observado no exemplo da Figura 6.11, são introduzidas 2 camadas de registradores ao longo do *array*, da mesma forma como apresentado nos multiplicadores *array* com m=2. Desta forma, são também necessários 3 ciclos de relógio para produzir o resultado final. Além disto, o aspecto comum a ambas as arquiteturas é o fato de que os registradores são inseridos na saída dos circuitos somadores responsáveis pela soma dos termos dos produtos parciais. Entretanto, como pode ser observado na Figura 6.11, no multiplicador Booth é também necessário introduzir registradores na saída dos circuitos codificadores para se ter a operação correta a cada ciclo de relógio.

6.1.5.4 Comparações de Desempenho

A Tabela 6.9 apresenta resultados de área, atraso, níveis lógicos e consumo de potência para os multiplicadores Booth Modificado e os multiplicadores *array* m=2, na versão *pipelined*.

TABELA 6.9 – Área, níveis lógicos, atraso e potência para multiplicadores paralelos de 16 bits em complemento de 2 na forma *pipelined*.

	Área	%	Atraso (ns)	%	Níveis Lógicos	%	Potência (mW)	%
Array Bin	5826	–	111,5	–	41	–	58,6	–
Array Hib	6468	+11,0	112,3	+0,7	42	+2,4	59,1	+0,9
Booth	5676	-14,4	128,5	+4,7	48	+17,0	69,6	+18,7

A introdução de registradores ao longo das camadas dos *arrays* aumenta a área das arquiteturas, comparadas às arquiteturas não *pipelined*. Entretanto, deve-se observar que a introdução de registradores ao longo dos circuitos reduz os seus caminhos críticos, tendo-se menores valores de atraso e níveis lógicos em relação às arquiteturas não *pipelined*, como pode ser comparado nas Tabelas 6.7 e 6.9.

Em relação às arquiteturas *pipelined*, observa-se que os multiplicadores Booth apresentam menores valores de área, como pode ser visto na Tabela 6.9, devido à menor complexidade dos módulos que processam os termos dos produtos. Entretanto, como pode ser observado nesta mesma tabela, estas arquiteturas apresentam maiores valores de atraso e níveis lógicos em relação às arquiteturas *array*. Este aspecto está relacionado à maior quantidade de circuitos presentes no caminho crítico da arquitetura Booth *pipelined*, que produz um maior número de interconexões. Nas Figuras 6.10 e 6.11, podem ser vistos os caminhos críticos das arquiteturas *array* e Booth respectivamente. Como pode ser observado na Figura 6.10, no multiplicador *array* o caminho crítico é dado por (1) um elemento multiplicador e um elemento somador do primeiro módulo do produto e (2) um elemento somador da linha de soma do produto parcial. Por outro lado, a Figura 6.11 mostra que em uma arquitetura Booth, o caminho crítico é dado por (1) circuito codificador, (2) um circuito para a geração do primeiro operando, que é composto por um multiplexador e meio somadores e (3) um circuito somador completo, responsável pela soma do primeiro e segundo operandos.

O menor número de níveis lógicos apresentado pelas arquiteturas *array* está de acordo com as observações feitas anteriormente, para as arquiteturas não *pipelined*, pois essas arquiteturas apresentam caminhos mais balanceados para os blocos básicos presentes no caminho crítico. Este aspecto contribui para aumentos em redução de potência por parte das arquiteturas *array pipelined*, devido à menor geração de transições espúrias, como mostrado na Tabela 6.9 para padrões de entrada senoidais. Como a arquitetura Booth apresenta uma menor complexidade em área, como mostrado na Tabela 6.9, esta arquitetura consegue reduzir mais potência quando comparada à versão não *pipelined*. Entretanto, os menores valores de atraso e níveis lógicos apresentados pelas arquiteturas *array* permite que estas arquiteturas continuem sendo mais eficentes em consumo de potência quando aplicado às entradas dos circuitos um padrão de sinal senoidal, como mostrado na Tabela 6.9.

Para sinais de padrão randômico, aplicados às entradas dos multiplicadores, onde o aspecto da correlação não está presente, tem-se um valor de redução de potência do multiplicador *array* Binário em relação ao multiplicador Booth, muito próximo ao resultado obtido para entradas senoidais, como pode ser visto nas Tabelas 6.9 e 6.10. Entretanto, como também pode ser comparado nestas duas tabelas, o aspecto da falta de correlação nas entradas dos multiplicadores, aumenta o consumo de potência da arquitetura *array* Híbrida em relação à arquitetura Binária.

TABELA 6.10 – Valores de potência para sinal de padrão randômico.

Arquiteturas	Potência(mW)	%
Array Binário(m=2)	100,4	–
Array Híbrido(m=2)	108,6	+ 8,1
Booth Modificado	123,2	+22,7

6.2 Multiplicação Desloca-Soma em Complemento de 2

Ao longo dos anos alguns trabalhos têm abordado a decomposição da multiplicação de constantes em implementações com operações desloca-soma em complemento de 2 [REI 60, HAR 89, CHA 93, POT 96, PAS 99, NGU 2000]. Alguns destes trabalhos propõem transformações nos circuitos para redução do número de deslocamentos e somas, tais como eliminação de subexpressões comuns, associatividade, distributividade e comutatividade. No nosso trabalho o principal aspeto é a exploração das operações em código Híbrido em complemento de 2, como será mostrado nesta parte do trabalho.

6.2.1 Trabalho Relacionado

Como dito anteriormente, o enfoque de alguns trabalhos relacionados com a decomposição das operações de multiplicação é a transformação do circuito para redução do número de deslocamentos e somas. Em [POT 94, PAS 99], por exemplo, busca-se o compartilhamento comum dos produtos para eliminação de subexpressões comuns. Em [POT 94], experimenta-se a redução no número de deslocamentos, adições e multiplicações em operações polinomiais. De acordo com os autores, tem-se um número médio de reduções de 54%, 56% e 15% respectivamente para estas operações. Em [PAS 99], o objetivo é identificar padrões múltiplos no conjunto de coeficientes de um filtro FIR e remover estes padrões. O principal alvo deste trabalho é a redução do número de circuitos somadores em coeficientes de filtro FIR. De acordo com os autores, a técnica utilizada apresentou uma significativa redução de *hardware* com satisfatório tempo de execução.

Em [NGU 2000], a abordagem envolve a decomposição de operações de multiplicação complexa em operações desloca-soma elementares. Isto é realizado através do máximo compartilhamento entre computações intermediárias. Outra abordagem deste trabalho envolve a transformação numérica do sistema original de modo a se obter novas arquiteturas com diferentes coeficientes para o termo multiplicador, de forma a facilitar o compartilhamento entre as computações. De acordo com os autores, estas transformações possibilitam uma redução em 30% do total do número

de operações.

Uma outra abordagem envolvendo transformações no modelo matemático do sistema é apresentado em [CHA 93]. Neste trabalho, uma técnica denominada *number-splitting* é utilizada para simplificar os coeficientes do sistema e desta forma, reduzir a complexidade das multiplicações. De acordo com os autores, com a aplicação desta técnica é possível uma redução em área de 36%. Entretanto, a redução em *hardware* produz o aumento no atraso em alguns circuitos.

Embora os trabalhos apresentados abordem diferentes técnicas de redução do número de operações desloca-soma, nenhum destes aborda operações considerando operandos em um código diferente do Binário. Em nosso trabalho, apresenta-se um circuito multiplicador desloca-soma em complemento de 2 que opera com operandos em código Híbrido. Como será mostrado, a facilidade de conversão entre os códigos Binário e Híbrido, mostrada nas seções anteriores, pode ser aplicada para implementação de arquiteturas de circuitos operando em código Híbrido com diferentes grupos de m bits. Os circuitos mostrados nesta parte do trabalho serão utilizados posteriormente em arquiteturas de filtros FIR dedicadas na forma totalmente paralela, onde os coeficientes são fixos, tendo-se um circuito multiplicador para cada estágio do filtro.

6.2.2 Operação Desloca-Soma em Código Híbrido

Para um circuito multiplicador desloca-soma Híbrido a melhor alternativa de implementação é utilizar um multiplicador Binário desloca-soma convencional com portas lógicas EXOR nas entradas e saídas do circuito, para transformação da representação dos dados. A estrutura desta operação é mostrada na Figura 6.12.

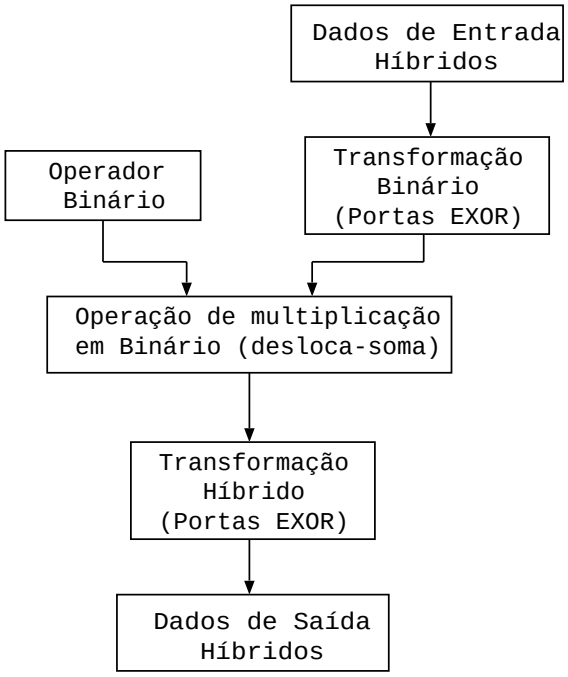

FIGURA 6.12 – Estrutura da operação de multiplicação desloca-soma em código Híbrido.

Como pode ser observado na Figura 6.12, a operação do multiplicador desloca-soma Híbrido mantém o operador em código Binário. Desta forma, as portas lógicas

EXOR nas entradas do circuito garantem a operação de multiplicação em Binário. Por outro lado, estas portas lógicas nas saídas do circuito garantem a mudança de representação de forma a se ter os dados disponíveis nas saídas em código Híbrido. Um exemplo da operação de multiplicação desloca-soma em código Híbrido é apresentado na Figura 6.13 para um número de 4 bits com grupos de bits m=2 e 4.

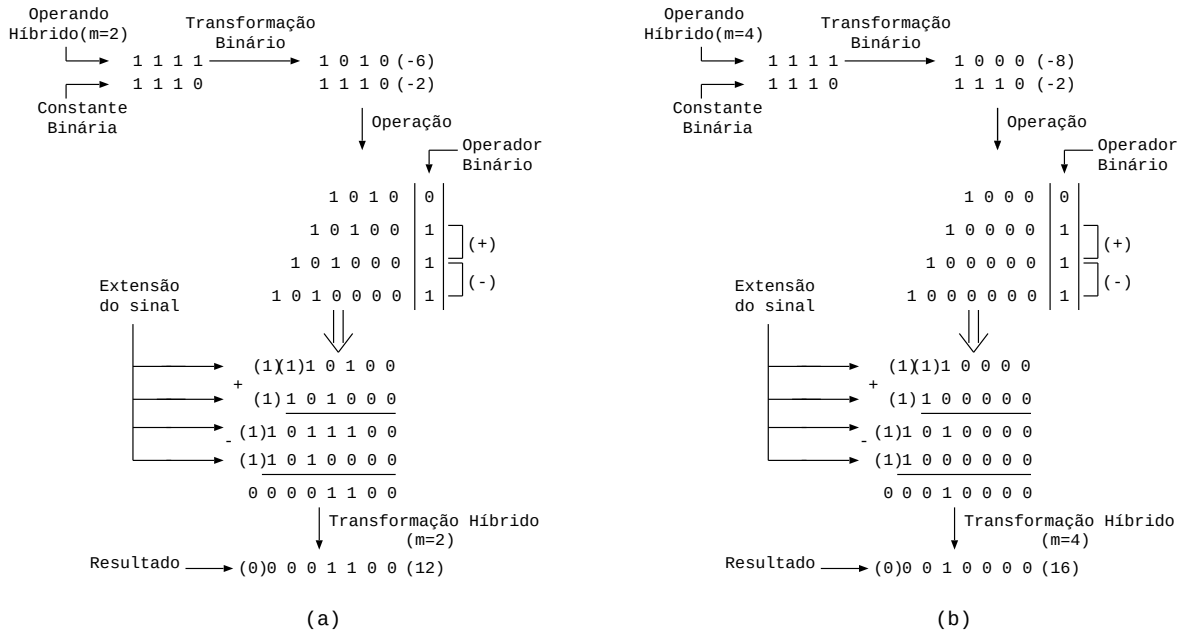

FIGURA 6.13 – Exemplos de multiplicação desloca-soma em código Híbrido com m=2 e 4.

Como se observa na Figura 6.13, a diferença entre as operações em código Híbrido com grupos de bits m=2 e 4 consiste na mudança de representação dos dados com código Gray nos grupos de bits. Entretanto, para ambos os casos se utiliza um multiplicador desloca-soma Binário em complemento de 2 com extensão do sinal, como pode ser observado nas Figuras 6.13(a) e 6.13(b). Naturalmente, a arquitetura do multiplicador Híbrido que envolve grupos de bits m=4 envolve uma maior complexidade devido à conversão entre os códigos Gray e Binário, sendo utilizada uma maior quantidade de portas lógicas EXOR, como mostrado na Figura 6.14(b).

Como se observa nas Figuras 6.14(a) e (b), para operações desloca-soma em código Híbrido com valores de m=2 e 4, utiliza-se o mesmo circuito multiplicador desloca-soma Binário. Neste circuito Binário são utilizados circuitos somadores ou subtratores de acordo com os bits do operador. De fato, para todos os bits do operador em nível lógico 1 são utilizados circuitos somadores que fazem as somas dos operandos deslocados à esquerda, menos o último bit do operador que define a utilização de um circuito subtrator, caso este bit esteja em nível lógico 1, que indica uma operação com sinal. As portas lógicas EXOR fazem as conversões dos dados entre os códigos Binário e Híbrido. Para uma palavra de dados de W bits e m=2, são necessárias $\frac{W}{2}$ portas nas entradas do multiplicador e W portas nas saídas do circuito para uma operação Híbrida. Para uma operação Híbrida, com m=4, são necesárias $3\frac{W}{4}$ portas lógicas EXOR nas entradas para conversão do código Híbrido para o Binário e o dobro desta quantidade para a conversão de Binário para Híbrido na saída do multiplicador. Como pode ser observado na Figura 6.14(b), as

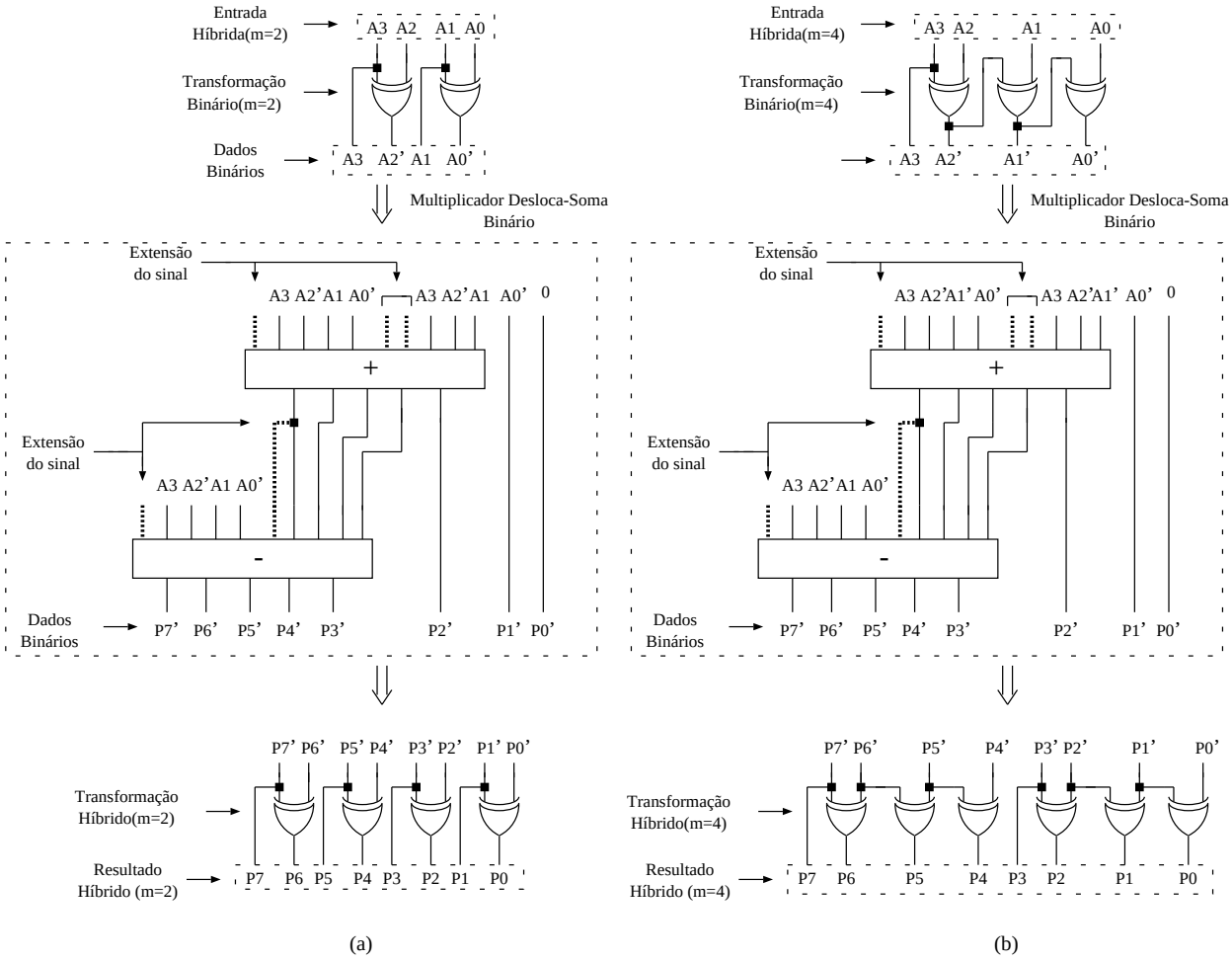

FIGURA 6.14 – Arquiteturas dos multiplicadores desloca-soma Híbridos com m=2 e 4.

conversões entre os códigos Híbrido e Binário nas entradas e saídas do multiplicador desloca-soma são realizadas de formas diferente. Estas conversões são realizadas como mostrado anteriormente nas Equações 4.3, 4.4, 4.5 e 4.6.

As arquiteturas de multiplicadores desloca-soma são normalmente utilizadas para operações de multiplicação de operandos com valores constantes. No próximo capítulo será mostrado o impacto da utilização destes multiplicadores Binário e Híbrido na arquitetura de um filtro FIR totalmente paralelo de 16 bits, onde para cada coeficiente do filtro é utilizada uma destas arquiteturas.

6.3 Resumo

Neste capítulo foram apresentadas novas arquiteturas de circuitos multiplicadores em códigos Binário e Híbrido na base 2^m que operam em números em complemento de 2. Foi mostrado que a estrutura destes multiplicadores mantém o mesmo nível de regularidade de um multiplicador *array* convencional (m=1). Foram apresentados resultados que mostram que as arquiteturas *array* propostas, operando em grupos de bits m=2, podem ser mais eficientes do que a arquitetura Booth Modificada na base 4. Também foi mostrado que é possível se obter redução de atraso e

potência para a arquitetura Binária com m=4. Estas reduções para multiplicadores Booth de bases maiores se torna difícil, devido à complexidade para codificação dos operandos em grupos maiores de bits. A regularidade das arquiteturas propostas permite a aplicação de outras técnicas de redução de potência. Neste contexto, foram apresentadas implementações das arquiteturas *array* e Booth Modificada na versão *pipelined*. Os resultados mostraram que mesmo na versão *pipelined*, a maior regularidade das arquiteturas *array* propostas permite um melhor desempenho e redução de potência. Outro aspecto que foi abordado neste capítulo é a possibilidade de utilização de circuitos multiplicadores desloca-soma em código Híbrido. Como foi apresentado, a operação envolve a utilização de circuitos Binários para deslocamento dos operandos e a utilização de portas lógicas EXOR nas entradas e saídas do circuito para conversão dos dados. A complexidade do circuito é menor para a operação Híbrida m=2, pois envolve uma menor quantidade de portas EXOR.

7 Exploração Arquitetural com Técnicas de Baixa Potência: Estudo de Casos

Como abordado no Capítulo 4, trabalhos iniciais de reestruturação da descrição de um circuito RTL para baixa potência foram propostos por Chandrakasan [CHA 95a], onde são utilizadas transformações para redução do número de computações e substituição de operações mais complexas, tais como multiplicações, por operações mais simples. Também é explorado o uso de *loop unrolling* com *pipelining* para permitir a redução da tensão da fonte de alimentação.

Algumas destas técnicas de transformação têm sido utilizadas na realização de filtros FIR, onde são apresentadas implementações em processadores digitais programáveis e arquiteturas dedicadas [MEH 95, MEH 96, SAN 97, MEH 98, MOL 98, ERD 2000, MUH 2000]. No caso das aplicações onde a flexibilidade do processador programável não é necessária, a implementação em arquiteturas dedicadas aparece como melhor alternativa de implementação, que resulta tipicamente em maiores desempenhos e menor potência [MEH 98a].

Para a implementação em arquiteturas dedicadas, o uso de transformações tem sido direcionado à velocidade de operação e complexidade computacional [MEH 98a]. Em relação às técnicas voltadas à velocidade de operação, o principal objetivo é a redução da tensão de alimentação a valores mínimos para a redução do consumo de potência. Neste contexto, técnicas de processamento paralelo e *pipelining* têm sido aplicadas para a implementação de filtros FIR como uma forma de operação em uma tensão mais baixa, resultando em redução de potência proporcional ao quadrado da tensão da fonte de alimentação.

Nesta parte do trabalho, propomos a aplicação de algumas das técnicas de transformação e em especial das técnicas que se direcionam ao aumento de desempenho e redução da atividade de chaveamento. Em particular, serão aplicadas transformações arquiteturais na implementação dos algoritmos de filtro FIR e FFT, levando em consideração aspectos de área, velocidade de operação e consumo de potência. Além disto, serão utilizadas as técnicas de baixa potência que levam em consideração as probabilidades de chaveamento do sinal. Desta forma, serão abordados os aspectos de utilização dos operadores aritméticos em códigos Binário e Híbrido na base 2^m nas arquiteturas de filtro FIR [COS 2002c] e FFT dedicadas.

7.1 Implementações de Filtro FIR

A operação de filtragem com resposta finita ao impulso (FIR) é realizada a partir do processo de convolução das amostras dos dados de entrada com a resposta ao impulso unitário do filtro. A saída $Y[n]$ de um filtro FIR de *N-tap* é dada pelo somatório dos pesos das últimas N amostras de dados de entrada, (onde $X[n]$ é a n-ésima amostra), conforme mostra a Equação 7.1.

$$Y[n] = \sum_{i=0}^{N-1} H[i]X[n-i] \tag{7.1}$$

As implementações apresentadas neste trabalho têm como alvo arquiteturas de 16 bits para um filtro FIR de 8ª ordem. São apresentadas arquiteturas de fil-

tros FIR totalmente paralelas e seqüenciais, sendo ambas em três versões: forma direta, *pipelined* e transposta [COS 2002c] e FFT. Adicionalmente, é apresentada uma arquitetura semi-paralela que melhora o desempenho do filtro em relação à arquitetura seqüencial pela duplicação do *hardware*, possiblitando que dois produtos parciais sejam realizados ao mesmo tempo.

As arquiteturas *pipelined* e forma transposta direta partem da idéia da técnica de *retiming* [LEI 83, CHA 95, MON 96a], ou seja, reposicionamento dos registradores para minimização da atividade de chaveamento, mantendo-se o comportamento funcional. A diferença consiste na latência do circuito que é maior na técnica *pipelining* utilizada neste trabalho. Uma abordagem similar da utilização da técnica *pipelining* na arquitetura totalmente paralela, é utilizada em [MUH 2000]. Neste trabalho, são utilizados circuitos multiplicadores *radix*-4 *Booth-encoded Wallace tree*. No nosso trabalho, as arquiteturas totalmente paralelas são compostas por circuitos multiplicadores tipo desloca-soma, descritos na Seção 6.2, como será abordado na próxima seção.

As arquiteturas semi-paralelas propostas neste trabalho partem da idéia da técnica *loop unrolling* onde duas amostras de saída são computadas em paralelo, baseadas em duas amostras de entrada. No nosso trabalho, as arquiteturas semi-paralelas computam duas amostras em paralelo, mas com uma saída disponível com a resposta do filtro a cada 4 ciclos de relógio para o filtro de 8ª ordem utilizado como estudo de caso. As implementações semi-paralelas utilizam aproximadamente o dobro do *hardware* apresentado por uma alternativa totalmente seqüencial e aproximadamente metade do *hardware* apresentado pela alternativa totalmente paralela, como será mostrado na próxima seção.

As implementações apresentadas nesta parte do trabalho foram realizadas no formato BLIF, na ferramenta SIS, conforme Seção 2.4.3. Os valores de área e velocidade de operação foram estimados no ambiente SIS. Os valores de consumo de potência para cada uma das arquiteturas são apresentados em função da energia gasta por cada amostra. Este parâmetro é dado de acordo com a Equação 7.2.

$$Energia\_por\_amostra = \frac{(Pot.media) \times (nT)}{n^o - de - amostras} \tag{7.2}$$

Na Equação 7.2, o parâmetro n corresponde ao número total de ciclos de relógio para processamento de todas as amostras. Para as arquiteturas mostradas nesta parte do trabalho é utilizado um sinal senoidal, portanto um valor fracionário entre -1 e 1, com uma quantidade de 10000 amostras. O parâmetro T representa o período mínimo de relógio para simulação. A potência média é calculada na ferramenta SLS a partir do valor do período T. Como o período utilizado foi o mesmo para todas as arquiteturas, este parâmetro pode ser suprimido da Equação 7.2, ficando esta reduzida à Equação 7.3, que passa a representar o consumo de potência normalizada por amostra, ou seja, a quantidade de energia gasta em um ciclo de relógio. O valor de período mínimo de relógio utilizado nas simulações representa o pior caso apresentado pela arquitetura totalmente paralela na forma direta, T=310ns. Para os barramentos de entrada e saída dos circuitos foi utilizado um valor de capacitância igual a 10fF, que corresponde ao menor valor de carga associado ao *fan-in* de uma porta. As máquinas de estado das arquiteturas das alternativas seqüencial e semi-paralela apresentam valores reduzidos de potência normalizada por amostra, não tendo desta forma, contribuição significativa para o aumento de potência global

nestas arquiteturas.

$$Potencia\_normalizada\_por\_amostra = \frac{(Pot.media) \times (n)}{n^o - de - amostras} \tag{7.3}$$

7.1.1 Arquiteturas Totalmente Paralelas

O método mais direto de implementação do filtro FIR é a forma direta totalmente paralela, onde registradores são utilizados como unidades de atraso entre os multiplicadores, como mostra a Figura 7.1(a). Nesta implementação o valor de entrada atual e as amostras anteriores são aplicados a cada entrada do multiplicador e as saídas dos multiplicadores são somadas juntas para formar a saída do filtro.

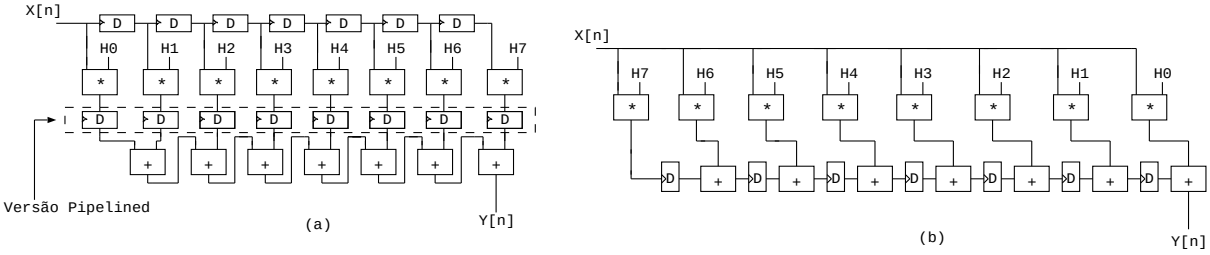

FIGURA 7.1 – Parte operativa das implementações de filtro FIR totalmente paralelas.

Na implementação do filtro FIR na forma direta, para cada ciclo de relógio são aplicados em simultâneo a todos os multiplicadores uma nova amostra, causando uma elevada taxa de atividade de chaveamento nos multiplicadores [ERD 96, ERD 2000]. Além disto, esta implementação pode produzir uma grande quantidade de *glitching*, causado pelas operações dos multiplicadores que se propagam pelos somadores. Em ambos os casos, a elevada taxa de atividade de chaveamento resulta em um elevado consumo de potência.

Neste trabalho, ambos os problemas são estudados a partir da implementação de duas arquiteturas totalmente paralelas chamadas *pipelined* e forma transposta, como mostrado na Figura 7.1(a) e na Figura 7.1(b) respectivamente. Na forma *pipelined*, um conjunto de registradores é introduzido entre as saídas dos multiplicadores e as entradas dos somadores, como mostrado na linha pontilhada da Figura 7.1(a). Na forma transposta, a posição dos registradores é alterada das entradas dos multiplicadores para as entradas dos somadores, como mostrado na Figura 7.1(b). Como pode ser observado na Figura 7.1, as implementações totalmente paralelas apresentam uma grande quantidade de circuitos multiplicadores. Desta forma, dependendo do tipo de operador de multiplicação utilizado nas arquiteturas, pode haver penalidade em seus parâmetros de área, consumo de potência e velocidade de operação. Ao longo dos anos, muitos pesquisadores têm investigado a decomposição de operações de multiplicação em eficientes implementações com deslocamentos e somas para redução de potência e otimização de *hardware* [NGU 2000]. É possível a utilização deste tipo de operador nas arquiteturas de filtro FIR totalmente paralelas, pelo fato de se ter associado a cada circuito multiplicador um coeficiente de valor fixo. Neste caso, o número de adições/subtrações usado para implementar as multiplicações dos coeficientes domina a complexidade do filtro [PAR 2001]. Neste

TABELA 7.1 – Valores área, período mínimo de relógio e potência normalizada por amostra para arquiteturas de filtro FIR totalmente paralelas.

Parâmetros	Direta	Transposta	Dif.(%) Transp→Direta	*Pipelined*	Dif.(%) *Pipe*→Direta
Area (Literais)	34282	34282	0	35828	+4,5
Período Mínimo de Relógio (ns)	307,3	271,7	-11,6	260,6	-15,2
Pot. normalizada por amostra	98,5	56,3	-42,7	60,2	-38,8

trabalho de pesquisa, são implementadas arquiteturas de filtros FIR totalmente paralelas utilizando circuitos multiplicadores desloca-soma com o objetivo de redução da atividade de chaveamento e atividade de *glitching* nas arquiteturas propostas.

Enquanto na alternativa *pipelined* os principais objetivos são a redução da atividade de *glitching* na saída dos circuitos multiplicadores e o aumento do desempenho (pelo aumento da freqüência de operação), na alternativa transposta o principal objetivo é reduzir a atividade de chaveamento na entrada dos somadores. A Tabela 7.1 mostra os resultados de área, período mínimo de relógio e consumo de potência normalizada por amostra para as alternativas arquiteturais totalmente paralelas.

Como pode ser observado pela Tabela 7.1, não há diferença de área entre as alternativas direta e transposta, pois a diferença entre estas duas arquiteturas consiste no deslocamento da posição dos registradores. Entretanto, a alternativa *pipelined* apresenta uma penalidade de área, pois são introduzidos registradores entre as saídas dos multiplicadores e as entradas dos somadores.

Em relação ao período mínimo de relógio, observa-se que a arquitetura na forma direta apresenta o maior valor entre as arquiteturas. Isto ocorre porque o caminho crítico desta arquitetura é realizado pelo caminho do primeiro multiplicador ligado à entrada, associado aos sete circuitos somadores que compõem o caminho até a saída. Para a arquitetura transposta, seis destes circuitos somadores são retirados do seu caminho crítico, sendo este estabelecido por um circuito multiplicador e um circuito somador. Desta forma, esta arquitetura apresenta menor valor de período mínimo de relógio em relação à arquitetura na forma direta, como mostrado na Tabela 7.1. Entretanto, a arquitetura *pipelined* apresenta o menor valor de período mínimo de relógio entre as arquiteturas. Isto ocorre porque nesta arquitetura o caminho crítico é dado apenas por um circuito multiplicador (que é superior à soma dos circuitos somadores em série).

Em relação ao consumo de potência normalizada por amostra, obtida a partir da Equação 7.3 com n=1, observa-se na Tabela 7.1 as alternativas *pipelined* e transposta com menores valores em relação à arquitetura na forma direta. Isto ocorre devido às configurações destas duas arquiteturas, que incluem registradores em pontos estratégicos do circuito com as funções de redução da atividade de *glitching* e redução da atividade de chaveamento nas entradas dos circuitos somadores, respectivamente. Deve-se observar entretanto, que devido ao fato da utilização de circuitos multiplicadores desloca-soma, a arquitetura transposta apresenta uma maior

redução de energia gasta por cada amostra em relação à arquitetura *pipelined*. Isto ocorre porque o circuito multiplicador desloca-soma não provoca a mesma quantidade de atividade de *glitching* que um circuito multiplicador tipo *array* por exemplo. Desta forma, para o tipo de alternativa arquitetural totalmente paralela com circuitos multiplicadores desloca-soma, a redução da atividade de chaveamento nas entradas dos somadores proporcionada pela arquitetura transposta é maior.

7.1.2 Arquiteturas Totalmente Seqüenciais

O segundo tipo de implementação mostrado neste trabalho, trata-se de arquiteturas totalmente seqüenciais, como uma maneira de reduzir os requerimentos de *hardware* para o algoritmo do filtro FIR, como mostrado na Figura 7.2.

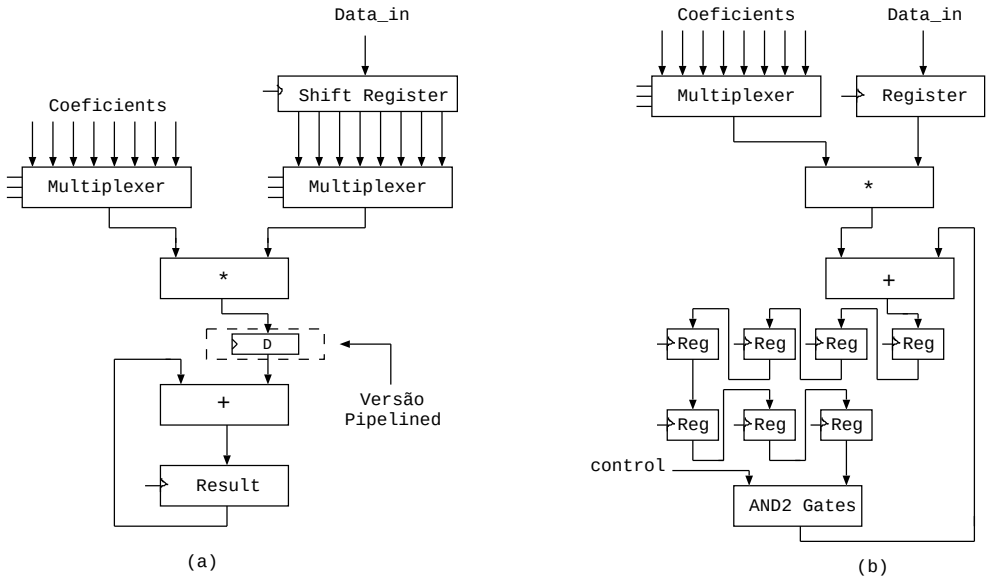

FIGURA 7.2 – Parte operativa das implementações seqüenciais do filtro FIR.

Em uma implementação seqüencial, a idéia básica é reduzir os requisitos de *hardware* pela reutilização do *hardware* tanto quanto possível. Neste caso, uma unidade de controle é utilizada no sentido de introduzir um número de passos de controle e cada produto da soma no filtro FIR é computado seqüencialmente e adicionado à soma acumulada no módulo acumulador. Neste trabalho, são implementados dois tipos de arquiteturas seqüenciais. Na primeira implementação, tem-se a arquitetura na forma direta, como mostrado na Figura 7.2(a), enquanto que na segunda implementação é apresentada a arquitetura na forma transposta, como mostrado na Figura 7.2(b). Na forma direta foi também projetada uma versão *pipelined* a partir da utilização de um registrador entre a saída do multiplicador e uma das entradas do somador, como mostrado na linha pontilhada na Figura 7.2(a). Na forma transposta são utilizados registradores para deslocamento do resultado da soma acumulada de uma amostra por cada um dos coeficientes. No bloco de portas lógicas AND uma entrada de controle garante que os resultados das somas acumuladas estejam disponíveis nos momentos corretos em uma das entradas do somador, para operação de soma com o resultado da multiplicação de um novo dado com os coeficientes.

TABELA 7.2 – Valores área, período mínimo de relógio e potência normalizada por amostra para arquiteturas de filtro FIR totalmente seqüenciais.

Parâmetros	Direta	Transposta	Dif.(%) Transp→Direta	*Pipelined*	Dif.(%) *Pipe*→Direta
Área (Literais)	5615	4786	-14,7	5808	+3,4
Período Mínimo de Relógio (ns)	282,6	282,6	0	272,6	-3,5
Pot. normalizada por amostra	337,73	290,49	-13,9	287,09	-14,9

Enquanto que nas arquiteturas totalmente paralelas é possível a utilização de circuitos multiplicadores tipo desloca-soma, por se ter um coeficiente fixo ligado a cada circuito multiplicador, para as alternativas seqüenciais e semi-paralelas não é possível a utilização deste operador, pois os coeficientes são colocados nas entradas dos mesmos circuitos multiplicadores. Desta forma, utiliza-se nesta arquitetura e na arquitetura semi-paralela os multiplicadores *array* propostos no âmbito deste trabalho. Nesta seção são apresentados resultados com a aplicação de multiplicadores *array* com m=1. Posteriormente, será mostrado o impacto da utilização dos operadores com maiores valores de m. Os valores de área, período mínimo de relógio e potência gasta por cada amostra para cada uma das arquiteturas são mostrados na Tabela 7.2.

A Tabela 7.2 mostra que a arquitetura transposta apresenta menor valor de área em relação às arquiteturas direta e *pipelined*. Isto ocorre, pois a forma transposta utiliza uma menor quantidade de circuitos multiplexadores na sua arquitetura. Na forma *pipelined* é utilizado um registrador na saída do circuito multiplicador e assim, esta arquitetura apresenta um maior valor de área em relação à arquitetura na forma direta.

Em relação ao período mínimo de relógio, observa-se que as arquiteturas na forma direta e transposta apresentam o mesmo valor, pois o caminho crítico destas arquiteturas é o mesmo sendo dado pelos circuitos multiplicador e somador. Entretanto, a arquitetura *pipelined* apresenta o menor valor para este parâmetro entre as arquiteturas, pois seu caminho crítico é dado apenas pelo circuito multiplicador.

A arquitetura *pipelined* também apresenta a maior redução de potência gasta por amostra entre as alternativas seqüenciais, como mostra a Tabela 7.2. Nesta tabela, estes valores foram obtidos usando n=8 na Equação 7.3. Para esta arquitetura, a redução da atividade de *glitching*, que é provocada pelo multiplicador tipo *array*, torna-se um ponto fundamental na redução da potência normalizada por amostra. A redução da atividade de *glitching* proporcionada pela arquitetura *pipelined* consegue ser mais eficiente do que a redução da atividade de chaveamento nas entradas do circuito somador verificada na arquitetura transposta. Embora na alternativa transposta a redução da atividade de chaveamento na entrada do circuito somador proporcione uma redução do consumo de potência normalizada por amostra em relação à arquitetura na forma direta, este valor não é significativo. Isto ocorre pelo fato da atividade de *glitching* se tornar mais intensa na arquitetura transposta devido ao uso do circuito multiplicador tipo *array*, o que minimiza o ganho

em redução de energia por amostra obtido na redução da atividade de chaveamento na entrada do circuito somador.

7.1.3 Arquiteturas Semi-Paralelas

Apesar da possibilidade de aumento de desempenho na arquitetura totalmente seqüencial pelo uso de *pipelining*, sua estrutura impõe uma restrição onde são necessários 8 ciclos de relógio para cada amostra. De outra forma, as arquiteturas totalmente paralelas podem operar de uma forma mais rápida, entretanto ao custo de uma maior quantidade de área devido à maior quantidade de *hardware* utilizada. No sentido de possibilitar uma alternativa intermediária entre as implementações totalmente paralela e totalmente seqüencial, é proposto neste trabalho uma arquitetura semi-paralela. Nesta arquitetura, os requisitos de *hardware* são duplicados, como mostra a Figura 7.3, e duas amostras podem ser processadas simultaneamente, aumentando o desempenho do filtro FIR devido ao fato de que o *hardware* pode operar em metade dos ciclos de relógio. Tal como para as restantes arquiteturas apresentadas anteriormente, são implementadas três versões da arquitetura semi-paralela, sendo estas implementações nas formas direta, *pipelined* e transposta. Na forma *pipelined* são utilizados dois registradores nas saídas dos circuitos multiplicadores como mostra a linha pontilhada da Figura 7.3(a). Na forma transposta, as somas acumuladas são deslocadas nos dois lados do circuito como mostra a Figura 7.3(b). Um circuito multiplexador auxiliar garante que os resultados das somas acumuladas do lado esquerdo da arquitetura estejam disponíveis nos momentos corretos em uma das entradas do circuito somador do lado direito da arquitetura.

Nas arquiteturas semi-paralelas, os coeficientes são compartilhados entre dois circuitos multiplicadores, sendo cada grupo de quatro coeficientes compartilhados por cada um destes circuitos. Desta forma, assim como nos circuitos seqüenciais, não é possível a utilização de circuitos multiplicadores tipo desloca-soma, sendo utilizado nas arquiteturas propostas circuitos multiplicadores tipo *array*. Os resultados de potência normalizada por amostra, área e período mínimo de relógio para estas arquiteturas são mostrados na Tabela 7.3.

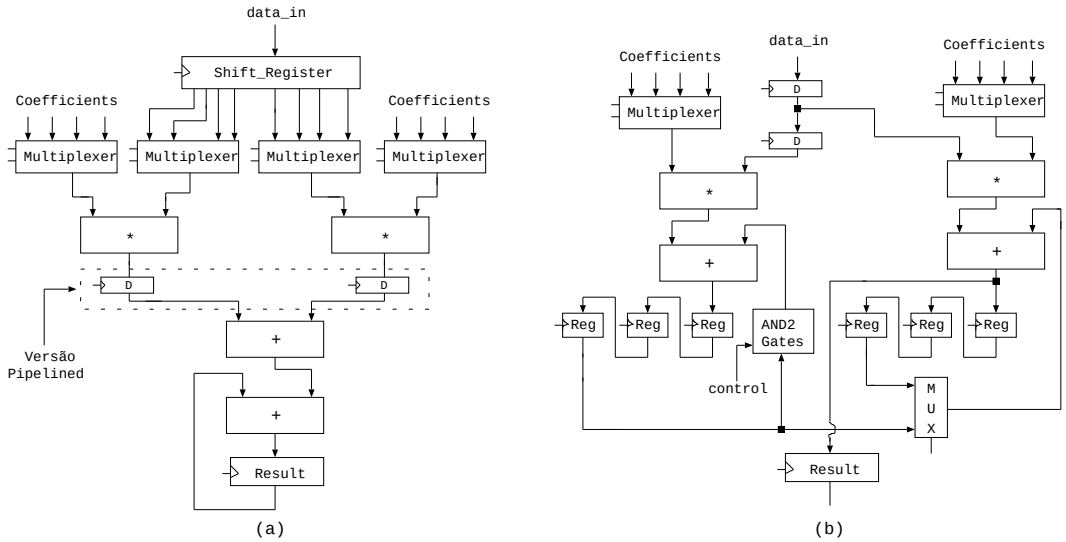

FIGURA 7.3 – Parte operativa das implementações semi-paralelas do filtro FIR.

TABELA 7.3 – Valores de área, período mínimo de relógio e potência normalizada por amostra para arquiteturas de filtro FIR semi-paralelas.

Parâmetros	Direta	Transposta	Dif.(%) Transp→Direta	*Pipelined*	Dif.(%) *Pipe*→Direta
Área (Literais)	9671	9225	-4,6	10055	+4
Período Mínimo de Relógio (ns)	287,4	281,1	-2,2	271,6	-5,5
Pot. normalizada por amostra	335,1	270,0	-19,4	268,1	-20.0

Da mesma forma como observado nas arquiteturas seqüenciais, verifica-se que na alternativa semi-paralela a arquitetura transposta apresenta o menor valor de área entre as arquiteturas, por utilizar menos circuitos multiplexadores. A arquitetura *pipelined* apresenta uma penalidade em área devido aos dois registradores utilizados nas saídas dos circuitos multiplicadores. Na implementação na forma direta, o caminho crítico é dado pelo caminho de um circuito multiplicador e dois circuitos somadores em série como mostrado na Figura 7.3. Na forma *pipelined*, os dois circuitos somadores em série não fazem parte do caminho crítico, sendo este estabelecido apenas por um circuito multiplicador. Desta forma, a arquitetura *pipelined* apresenta um menor valor de período mínimo de relógio em relação à arquitetura na forma direta como mostrado na Tabela 7.3. Na forma transposta, o caminho crítico é dado por um circuito multiplicador e um circuito somador. Desta forma, esta arquitetura apresenta um valor de período mínimo de relógio menor em relação à forma direta. Entretanto, comparado à arquitetura *pipelined*, a arquitetura transposta apresenta um maior valor de período mínimo de relógio como mostrado na Tabela 7.3, por apresentar um circuito a mais no seu caminho crítico (circuito somador).

A Tabela 7.3 também mostra que as arquiteturas transposta e *pipelined* conseguem reduções de potência normalizada por amostra, em relação à arquitetura na forma direta. Nesta tabela, estes valores foram obtidos usando n=4 na Equação 7.3. Isto ocorre pelo fato destas arquiteturas reduzirem a atividade de chaveamento nas entradas dos multiplicadores e atividade de *glitching* respectivamente. Deve-se observar que pelo fato das arquiteturas semi-paralelas utilizarem dois circuitos multiplicadores tipo *array*, a atividade de *glitching* se torna mais intensa nestas arquiteturas. Neste caso, a alternativa *pipelined* consegue uma maior redução de energia por apresentar os registradores nas saídas dos multiplicadores o que reduz significativamente a atividade de *glitching* como mostra a Tabela 7.3. A atividade de *glitching* mais intensa na alternativa transposta devido ao uso de dois circuitos multiplicadores tipo *array*, minimiza o ganho em redução de potência normalizada por amostra obtido com a redução da atividade de chaveamento nas entradas dos circuitos multiplicadores. Entretanto, a utilização destes dois circuitos multiplicadores proporciona alguma correlação nesta alternativa arquitetural, pois aplica-se o mesmo valor às entradas destes circuitos. Desta forma, a alternativa transposta apresenta uma redução de potência normalizada por amostra mais significativa em relação à forma direta em relação ao valor apresentado na alternativa seqüencial.

7.1.4 Comparações das Arquiteturas

Como observado na seção anterior, um conjunto de melhores alternativas de implementação do algoritmo de filtro FIR pode ser obtido a partir da exploração arquitetural. De acordo com o tipo de aplicação, podem ser utilizadas arquiteturas mais rápidas ou mais lentas. Se a aplicação requer uma maior velocidade de operação do *hardware*, então deve ser utilizada a arquitetura totalmente paralela. Neste caso, a arquitetura *pipelined* pode melhorar o desempenho do circuito. Na alternativa totalmente paralela, a arquitetura na forma transposta apresenta maior redução de potência normalizada por amostra. Entretanto, o valor de potência normalizada por amostra da arquitetura *pipelined* pode ser reduzido, pela redução da tensão de alimentação para manutenção do mesmo *throughput*. Neste caso, a arquitetura *pipelined* pode se tornar a melhor escolha para utilização na implementação totalmente paralela.

A arquitetura *pipelined* também se mostra mais eficiente em termos de velocidade de operação e redução de potência normalizada por amostra para as alternativas seqüencial e semi-paralela, pois além de apresentar um menor caminho crítico nestas alternativas arquiteturais, ainda reduz a quantidade de *glitching* provocada pelos circuitos multiplicadores.

Neste trabalho, tem-se utilizado a arquitetura semi-paralela como uma forma alternativa entre as implementações totalmente paralela e totalmente seqüencial. Este tipo de implementação pode ser uma boa alternativa para melhores resultados de área em relação às arquiteturas totalmente paralelas, e para melhores resultados de energia por amostra em relação às arquiteturas seqüenciais. Em relação ao período mínimo de relógio, observa-se que as arquiteturas seqüenciais e semi-paralelas apresentam praticamente os mesmos valores, visto que estas alternativas arquiteturais apresentam o mesmo caminho crítico. Entretanto, na forma direta, a arquitetura semi-paralela apresenta maior valor em relação à arquitetura seqüencial, por apresentar um circuito somador a mais em série. Como pôde ser observado nas Tabelas 7.1, 7.2 e 7.3, as alternativas seqüencial e semi-paralela apresentam maiores valores de período mínimo de relógio devido ao fato destas arquiteturas utilizarem multiplicadores tipo *array*.

As arquiteturas semi-paralelas podem ter os seus valores de consumo de potência normalizada por amostra reduzidos em relação às arquiteturas seqüenciais. Isto ocorre pelo fato das arquiteturas semi-paralelas poderem operar no dobro da freqüência de relógio e desta forma podem ter reduzidas as suas tensões de alimentação pela metade para manutenção do mesmo *throughput* em relação às arquiteturas seqüenciais. Como a energia apresenta um relacionamento quadrático com a tensão de alimentação (Equação 2.16), as arquiteturas semi-paralelas podem ter seus valores de potência normalizada por amostra reduzidos por um fator de 4, pela redução de V_{dd} pela metade, como mostram os resultados da Tabela 7.4. Observa-se pela Tabela 7.4 que a arquitetura *pipelined* apresenta o menor valor de potência normalizada por amostra. Isto ocorre porque esta arquitetura apresenta menores valores de potência normalizada por amostra nas alternativas seqüencial e semi-paralela, como mostrado nas Tabelas 7.2 e 7.3.

Embora a alternativa semi-paralela apresente uma redução considerável de potência normalizada por amostra pela redução da sua tensão de alimentação, observa-se que para a implementação do algoritmo de filtro FIR a alternativa totalmente paralela utilizando multiplicadores tipo desloca-soma é recomendável para

TABELA 7.4 – Comparação de potência normalizada por amostra entre arquiteturas de filtro FIR seqüencial e semi-paralela com redução de V_{dd}.

Arquiteturas	Pot. p/ amostra Seqüencial	Pot. p/ amostra Semi-Paralela com $V_{dd}/2$	Dif(%) Semi-Par vs. Seq
Direta	337,7	335,0/4 = 83,7	-253,9
Transposta	290,4	269,9/4 = 67,4	-223,0
Pipelined	287,0	268,0/4 = 67,0	-220,0

aplicações de baixa potência. Como pode ser observado, esta alternativa arquitetural apresenta um menor consumo de potência normalizada por amostra entre as três alternativas arquiteturais. Isto ocorre pelo fato desta alternativa utilizar circuitos multiplicadores desloca-soma em suas arquiteturas, que apresentam uma menor quantidade de *glitching* em relação aos circuitos multiplicadores tipo *array* utilizados nas alternativas seqüencial e semi-paralela. Além disto, nestas alternativas os valores dos coeficientes são fixos. As arquiteturas totalmente paralelas, além de apresentarem menores valores de potência normalizada por amostra, também apresentam menores valores de período mínimo de relógio entre as alternativas arquiteturais mostradas. Desta forma, o consumo de potência normalizada por amostra das arquiteturas desta alternativa pode proporcionar uma redução de energia adicional, a partir da redução da tensão de alimentação para manutenção do mesmo *throughput* em relação às demais alternativas arquiteturais (seqüencial e semi-paralela). Entretanto, observa-se um relacionamento de redução de energia ao custo de penalidade em área, pois as arquiteturas da alternativa totalmente paralela apresentam maiores valores de área comparados às demais alternativas arquiteturais.

7.1.5 Aplicação de Operadores Aritméticos na Base 2^m

Como apresentado na seção anterior, a estrutura *pipelined* apresenta um melhor desempenho nas diferentes alternativas arquiteturais implementadas. Desta forma, nesta parte do trabalho será abordada a utilização dos operadores aritméticos na base 2^m, apresentados no Capítulo 6, nas arquiteturas de filtro FIR na forma *pipelined*. Serão apresentados resultados de área, período mínimo de relógio e consumo de potência normalizada por amostra das arquiteturas totalmente paralela, totalmente seqüencial e semi-paralela com estes operadores. Os resultados são comparados com as alternativas arquiteturais utilizando m=1. As arquiteturas totalmente paralelas utilizam os multiplicadores desloca-soma apresentados no Capítulo 6, enquanto que as arquiteturas totalmente seqüencial e semi-paralela utilizam os multiplicadores *array* apresentados neste mesmo capítulo.

7.1.5.1 Análise de Resultados de Área

A arquitetura de filtro FIR utiliza operadores de soma e multiplicação para o cálculo da soma ponderada das N amostras de entrada como mostrado na Equação 7.1. Desta forma, dependendo do tipo de operadores aritméticos utilizados para realização do cálculo da soma ponderada dos sinais de entrada, pode haver diferentes valores de área na implementação da arquitetura específica para esta finalidade. Neste trabalho, os valores relativos de área para as arquiteturas de filtro FIR imple-

mentadas são estimados no ambiente SIS a partir do cálculo do número de literais de cada arquitetura. A Tabela 7.5 mostra os resultados de área para as alternativas arquiteturais totalmente paralela, seqüencial e semi-paralela na forma *pipelined* com a utilização de operadores aritméticos na base 2^m. Os valores percentuais apresentados nesta tabela, representam comparações relativas às arquiteturas com operadores Binários usando m=1.

TABELA 7.5 – Valores de área para alternativas arquiteturais na forma *pipelined*.

Alternativas Arquiteturais	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Totalmente Paralela	m=1	35828	-	-	-
	m=2	35828	0	35908	+0,2
	m=4	35828	0	36008	+0,5
Totalmente Seqüencial	m=1	6983	-	-	-
	m=2	7329	+4,9	8163	+16,8
	m=4	13199	+89,0	25188	+260,7
Semi-Paralela	m=1	11752	-	-	-
	m=2	12437	+5,8	14105	+20,0
	m=4	24177	+105,7	48155	+309,7

Como dito anteriormente, na alternativa arquitetural totalmente paralela são utilizados circuito multiplicadores desloca-soma para a operação do filtro FIR. Desta forma, os maiores de valores de área apresentados com operadores Híbridos, apresentados na Tabela 7.5, deve-se à utilização de portas lógicas EXOR nos barramentos de entrada/saída para codificação/decodificação dos dados. Desta forma, a área da arquitetura totalmente paralela em código Híbrido é dada pela soma da área da arquitetura em código Binário com as portas EXOR utilizadas nos barramentos de entrada/saída. Naturalmente, o valor de área é maior para a arquitetura Híbrida com m=4, devido ao maior número de portas EXOR utilizado para a conversão dos dados.

Como pode ser observado na Tabela 7.5, a arquitetura totalmente paralela com operadores Binários apresenta os mesmos valores para m=1, 2 e 4. Isto ocorre devido ao fato dos multiplicadores desloca-soma Binários com estes grupos de bits serem calculados da mesma forma, ou seja, não é necessário nenhum *hardware* adicional para codificação/decodificação dos dados. Esse aspecto não é verificado para as alternativas seqüencial e semi-paralela, onde são utilizados mutliplicadores *array*. Para estas arquiteturas, utiliza-se um multiplicador *array* para cada grupo de bits. Naturalmente, as arquiteturas seqüencial e semi-paralela utilizando maiores valores de m apresentam maiores valores de área. Isto devido ao fato dos módulos básicos dos circuitos multiplicadores aumentarem a complexidade com o aumento do valor de m.

A Tabela 7.5 mostra que as arquiteturas na forma *pipelined* seqüencial e semi-paralela com operadores aritméticos em código Híbrido apresentam maiores valores de área em relação às arquiteturas com operadores em código Binário. Esta diferença de área é mais significativa nestas alternativas arquiteturais, pelo fato dos operadores de multiplicação e soma em código Híbrido apresentarem maiores valores de área, como mostrado no Capítulo 6.

7.1.5.2 Análise de Resultados de Período Mínimo de Relógio

Apesar das arquiteturas *pipelined* do filtro FIR com operadores aritméticos na base 2^m com valores de m=2 e 4 apresentarem maiores valores de área, estas arquiteturas podem apresentar menores valores de período mínimo de relógio em relação às arquiteturas com operadores Binários com m=1, como mostram os resultados da Tabela 7.6. Como pode ser observado nesta tabela, para a arquitetura na forma *pipelined* totalmente paralela as arquiteturas Binárias apresentam os mesmos valores de atraso, pelo fato destas arquiteturas não precisarem de nenhum *hardware* adicional para codificação/decodificação dos dados, e desta forma, o caminho crítico é o mesmo para estas arquiteturas. Para a arquitetura totalmente paralela Híbrida com m=2, observa-se um pequeno acréscimo em relação à arquitetura Binária. Isto ocorre devido ao pequeno número de diferentes circuitos presentes no caminho crítico destas arquiteturas. Enquanto o caminho crítico da arquitetura Binária é composto por apenas um circuito multiplicador tipo desloca-soma, o caminho crítico da arquitetura Híbrida com m=2 é composto por este mesmo circuito multiplicador acrescido de portas lógicas EXOR no barramento de entrada. Para a arquitetura totalmente paralela com operador Híbrido com m=4, a quantidade de portas lógicas EXOR no barramento de entrada é maior. Desta forma, a arquitetura com este operador apresenta um valor significativamente maior de atraso, como mostrado na Tabela 7.6. Outro fator de acréscimo do atraso na arquitetura com operador Híbrido com m=4 é o maior caminho crítico envolvido no processo de conversão do código Gray para o código Binário, como mostrado anteriormente na Figura 6.14. Nesta conversão, a entrada de uma porta lógica EXOR depende do resultado de saída da porta EXOR anterior.

Para as arquiteturas *pipelined* seqüencial e semi-paralela, o caminho crítico é dado apenas por um circuito multiplicador, como mostrado nas Figuras 7.2 e 7.3, sendo este operador do tipo *array*. Como visto no Capítulo 6, o multiplicador *array* Binário na base 2^m apresenta o menor valor de atraso para m=4. Logo, as arquiteturas de filtro FIR seqüencial e semi-paralela com este operador apresentam os menores valores de atraso entre as arquiteturas, como mostrado na Tabela 7.6. Nestas alternativas arquiteturais, a utilização dos operadores Híbridos com m=2 apresentam valores de atraso inferiores às arquiteturas com operadores Binários com m=1, devido ao menor caminho crítico apresentado pelo operador Híbrido com este valor de m. Entretanto, como foi mostrado no Capítulo 6, os multiplicadores Híbridos com m=2 apresentam um pequeno maior valor de atraso em relação ao multiplicador Binário com este mesmo valor de m. Desta forma, as arquiteturas seqüencial e semi-paralela com operadores Híbridos com m=2 apresentam um pequeno maior valor de atraso. Esta diferença aumenta para arquiteturas com operadores Híbridos com m=4. Neste caso, a maior complexidade apresentada pelo multiplicador Híbrido com este valor de m faz com que as arquiteturas seqüencial e semi-paralela com estes operadores apresentem os maiores valores de atraso entre as arquiteturas, como mostrado na Tabela 7.6.

7.1.5.3 Análise de Resultados de Potência por Amostra

A aplicação direta dos operadores em código Híbrido nas arquiteturas *pipelined* revelam um pequeno acréscimo de consumo de potência normalizada por amostra em relação às arquiteturas com operadores em código Binário. Na arquitetura total-

TABELA 7.6 – Valores de atraso em ns para alternativas arquiteturais na forma *pipelined*.

Alternativas Arquiteturais	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Totalmente Paralela	m=1	260,6	-	-	-
	m=2	260,6	0	278,0	+6,6
	m=4	260,6	0	317,8	+21,9
Totalmente Seqüencial	m=1	296,3	-	-	-
	m=2	244,8	-17,3	250,1	-15,6
	m=4	234,2	-20,9	288,8	-2,5
Semi-Paralela	m=1	294,3	-	-	-
	m=2	242,8	-17,5	248,1	-15,7
	m=4	232,2	-21,1	286,8	-2,5

TABELA 7.7 – Valores de potência normalizada por amostra para alternativas arquiteturais na forma *pipelined*

Alternativas Arquiteturais	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Totalmente Paralela	m=1	56,37	-	-	-
	m=2	56,37	0	58,69	+4,1
	m=4	56,37	0	60,72	+7,7
Totalmente Seqüencial	m=1	290,49	-	-	-
	m=2	155,73	-46,4	180,68	-37,8
	m=4	128,97	-55,6	228,56	-21,3
Semi-Paralela	m=1	269,08	-	-	-
	m=2	136,89	-49,1	158,27	-41,2
	m=4	112,85	-58,0	215,03	-20,1

mente paralela como a operação em código Híbrido é realizada a partir da utilização da arquitetura Binária com portas EXOR adicionais nas entradas e saídas do circuito, tem-se um menor acréscimo de consumo de potência normalizada por amostra, como mostrado na Tabela 7.7. Com valores de m=1, 2 e 4, as arquiteturas totalmente paralelas em código Binário apresentam os mesmos valores de potência normalizada por amostra, pois nesta alternativa arquitetural são utilizados circuitos multiplicadores desloca-soma que não apresentam diferenças em código Binário em diferentes valores de m.

Nas arquiteturas seqüencial e semi-paralela, a operação é realizada a partir da reutilização do *hardware* tanto quanto possível. Nestas alternativas, os circuitos multiplicadores *array* são os maiores responsáveis pelo consumo de potência nas arquiteturas de filtros FIR. Neste caso, para as alternativas seqüencial e semi-paralela que utilizam este tipo de multiplicador, os circuitos com operadores aritméticos com valores de m=2 e 4 apresentam menores valores de potência em relação às arquiteturas com valores de m=1 nos códigos Binário e Híbrido, como pode ser observado na Tabela 7.7. Isto devido à menor profundidade lógica e menores valores de atraso

apresentados pelas arquiteturas com operadores que utilizam valores de m=2 e 4. Pelo fato dos circuitos seqüencial e semi-paralelo com operadores Binários com m=4 apresentarem os menores valores de atraso entre as arquiteturas, como mostrado anteriormente na Tabela 7.6, estas arquiteturas apresentam os menores valores de potência normalizada por amostra nestas alternativas arquiteturais, como pode ser observado na Tabela 7.7.

Deve-se observar que nas alternativas seqüencial e semi-paralela, não há correlação dos dados nas entradas dos circuitos multiplicadores. Além disto, tem-se que nos circuitos somadores a operação em código Híbrido apresenta maior consumo de potência, visto que este circuito é composto pelo somador Binário adicionado de portas lógicas EXOR nas entradas e saídas do circuito, como mostrado na Seção 5.1.2.1. Desta forma, a falta de correlação dos dados nas entradas dos circuitos multiplicadores associado ao aspecto do maior consumo de potência apresentado nos somadores em código Híbrido faz com que as arquiteturas seqüencial e semi-paralela com estes operadores, para valores de m=2 e 4, apresentem um maior acréscimo de potência normalizada por amostra em relação às arquiteturas com operadores Binários com estes mesmos valores de m, como mostrado na Tabela 7.7. Este aspecto se torna particularmente mais expressivo nas arquiteturas com m=4 do que nas arquiteturas com m=2. Isto devido ao fato da conversão entre os códigos Gray e Binário nos barramentos de dados ser mais complexa com m=4.

7.1.6 Comparação de Arquiteturas com Multiplicador Booth

Como apresentado no capítulo anterior, os multiplicadores *array* propostos neste trabalho e o multiplicador Booth apresentam em comum a possibilidade de redução das linhas de produtos parciais fazendo com que estas arquiteturas apresentem valores reduzidos de potência. A Tabela 7.8 mostra os resultados de potência normalizada por amostra para arquiteturas seqüencial e semi-paralela na versão *pipelined* com os multiplicadores *array* propostos neste trabalho e multiplicador Booth, com as arquiteturas operando na base 4.

TABELA 7.8 – Valores de potência normalizada por amostra para arquiteturas seqüencial e semi-paralela com multiplicadores na base 4.

Alternativas Arquiteturais	Booth	Binário	Diferença (%) Bin vs. Booth	Híbrido	Diferença (%) Hib vs. Booth
Seqüencial	215,4	155,7	-27,7	180,7	-16,1
Semi-Paralela	188,6	136,9	-27,4	158,3	-16,1

Como pode ser observado na Tabela 7.8, as arquiteturas seqüencial e semi-paralela com multiplicadores *array* nos códigos Binário e Híbrido apresentam menores valores de potência normalizada por amostra em relação às arquiteturas com multiplicador Booth. De fato, visto que os multiplicadores são responsáveis pela maior parte do consumo de potência das arquiteturas dos filtros FIR, tem-se que os multiplicadores *array* conseguem um maior ganho de potência pela sua estrutura mais simples e menores valores de atraso e caminho crítico, como foi apresentado no capítulo anterior. Desta forma, a utilização das arquiteturas *array* em circuitos da área DSP, como os filtros FIR mostrados neste capítulo, pode se tornar uma boa

alternativa para projetos de baixa potência.

7.2 Implementações da Transformada Rápida de Fourier

A transformada rápida de Fourier (FFT) é uma classe de algoritmos eficientes para processamento da transformada discreta de Fourier (DFT). Os algoritmos FFT são projetados tipicamente para minimizar o número de multiplicações e adições, mantendo-se uma forma simples em sua estrutura.

Os algoritmos FFT efetuam o cálculo da DFT de uma forma mais rápida a partir da simplificação matemática e classificação da seqüência de valores de entrada. O número de operações envolvido no cálculo da FFT depende da complexidade do tipo de algoritmo utilizado.

Na estrutura computacional de uma FFT, uma borboleta representa a parte central de cálculo de cada estágio. A complexidade da borboleta está associada ao tipo de algoritmo utilizado para o cálculo da FFT. Neste trabalho, o enfoque é a implementação de arquiteturas do algoritmo FFT de fator comum na base 2 com decimação em freqüência. No Capítulo 5, Figura 5.16, foi mostrada a estrutura padrão de uma borboleta para o cálculo da FFT com este algoritmo e, como pode ser visto nesta figura, a borboleta efetua o cálculo de termos complexos de acordo com as Equações 7.4 e 7.5.

$$C_{complexo} = A_{complexo} + B_{complexo} \tag{7.4}$$

$$D_{complexo} = (A_{complexo} - B_{complexo}) \times W_{complexo} \tag{7.5}$$

Desta forma, na estrutura de uma borboleta para o algoritmo FFT de fator comum na base 2 com decimação em freqüência são efetuadas uma soma complexa, uma subtração complexa e uma multiplicação complexa. A forma estendida destas operações complexas, considerando os termos real e imaginário das amostras de dados e dos coeficientes (*twiddle factors*), é mostrada nas equações abaixo, onde $X_{complexo} = X_r + iX_i$.

$$C_r = A_r + B_r \tag{7.6}$$

$$C_i = A_i + B_i \tag{7.7}$$

$$D_r = (A_r - B_r) \times W_r - (A_i - B_i) \times W_i \tag{7.8}$$

$$D_i = (A_i - B_i) \times W_r + (A_r - B_r) \times W_i \tag{7.9}$$

A Figura 7.4 mostra a estrutura dos operadores aritméticos necessários para a composição da borboleta, considerando os termos real e imaginário.

Como deve ser observado na Figura 7.4, embora sejam apresentados 5 circuitos subtratores na estrutura da borboleta, 2 destes são partilhados pelas entradas reais e outros 2 são partilhados pelas entradas imaginárias.

As implementações do algoritmo FFT mostradas nesta parte do trabalho utilizam como base a estrutura da borboleta mostrada na Figura 7.4 e têm como alvo

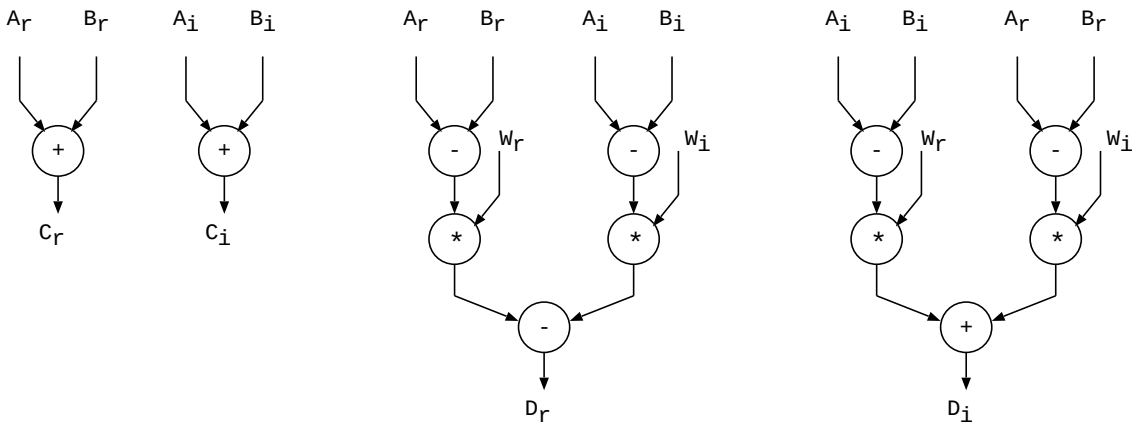

FIGURA 7.4 – Estrutura dos operadores aritméticos para a borboleta utilizada no algoritmo FFT de fator comum com decimação em freqüência na base 2.

arquiteturas de 16 bits para uma FFT de 16 pontos. São apresentadas arquiteturas FFT nas versões totalmente seqüencial e semi-paralela, sendo ambas na forma direta e na forma *pipelined*. Para o algoritmo FFT, uma implementação totalmente paralela não se torna atrativa devido a grande quantidade de *hardware* envolvida. Para este tipo de implementação, com 16 pontos são necessárias 32 borboletas operando simultaneamente, o que representa um total de 512 operadores aritméticos (96 circuitos somadores, 96 circuitos subtratores e 128 circuitos multiplicadores). Por esta razão, este tipo de implementação não é abordado neste trabalho.

Na forma *pipelined* a idéia central é a redução do caminho crítico e da atividade de *glitching* nas arquiteturas, pela introdução de registradores nas saídas dos circuitos multiplicadores. Esta operação aumenta a latência das arquiteturas em um ciclo de relógio. Ao longo dos anos diversos trabalhos têm abordado a implementação de arquiteturas FFT na forma *pipelined* em processadores dedicados [SZW 94, GOR 95, LI 99, YU 2000] para aplicações em tempo real com *throughput* elevado. A idéia da maior parte destes trabalhos é manter a regularidade das arquiteturas para uma fácil implementação VLSI, além da maior velocidade de processamento das operações aritméticas.

No nosso trabalho, explora-se a maior velocidade de operação da arquitetura FFT a partir da implementação de uma arquitetura semi-paralela. Nesta arquitetura, a FFT opera na metade dos ciclos de relógio utilizados para a arquitetura seqüencial, pois são utilizadas duas borboletas para processamento de quatro amostras simultaneamente. Na arquitetura semi-paralela, são implementadas arquiteturas na forma direta e na forma *pipelined*. As arquiteturas FFT mostradas nesta seção utilizam o multiplicador *array* em complemento de 2 com m=1 apresentado no Capítulo 5. Posteriormente, serão apresentados resultados de área, período mínimo de relógio e consumo de potência em arquiteturas com operadores aritméticos com valores de m=2 e 4.

O consumo de potência das arquiteturas é estimado em potência normalizada por transformada e é calculado de acordo com a Equação 7.10. Esta equação é semelhante à Equação 7.3, com a diferença que em uma transformada estão envolvidas N amostras. Nesta equação, o termo n representa o número total de ciclos de relógio gastos por cada arquitetura. Na Equação 7.10, a potência média é estimada na ferramenta SLS, utilizando-se um sinal senoidal de 10000 pontos, portanto um valor

fracionário entre -1 e 1, como vetor de entrada, realizando-se portanto, 625 transformadas para arquiteturas de 16 pontos ($\frac{10000}{16}$). As simulações das arquiteturas foram realizadas utilizando um período mínimo de relógio igual a 500ns que representa o pior caso de atraso para a arquitetura semi-paralela na forma direta com operador Híbrido e valor de m=4. O valor de atraso das arquiteturas foi estimado no ambiente SIS com modelo de atraso geral da bilbioteca `mcnc`. Os valores de área são também estimados no ambiente SIS e são fornecidos em termos da quantidade de literais.

$$Potencia\_normalizada\_por\_transformada = \frac{(Pot.media) \times (n)}{n^o - de - amostras} \times N \qquad (7.10)$$

7.2.1 Arquiteturas Seqüenciais

Em uma implementação do algoritmo FFT, a borboleta tem um papel central no cálculo de cada tranformada que é processada em cada estágio do fluxo de dados. Na arquitetura seqüencial, a borboleta utiliza quatro circuitos multiplicadores que efetuam a multiplicação dos dados com apropriados coeficientes, como mostrado na Figura 7.4. Nesta figura, os coeficientes reais e imaginários são representados por W_r e W_i respectivamente. Os coeficientes são armazenados em memórias ROM de tamanho $N/2$. Nesta seção será mostrada a implementação de uma arquitetura do algoritmo FFT totalmente seqüencial que utiliza apenas uma borboleta para o processamento da FFT. A parte operativa desta arquitetura é mostrada na Figura 7.5, onde não está representada a ROM para armazenamento dos coeficientes.

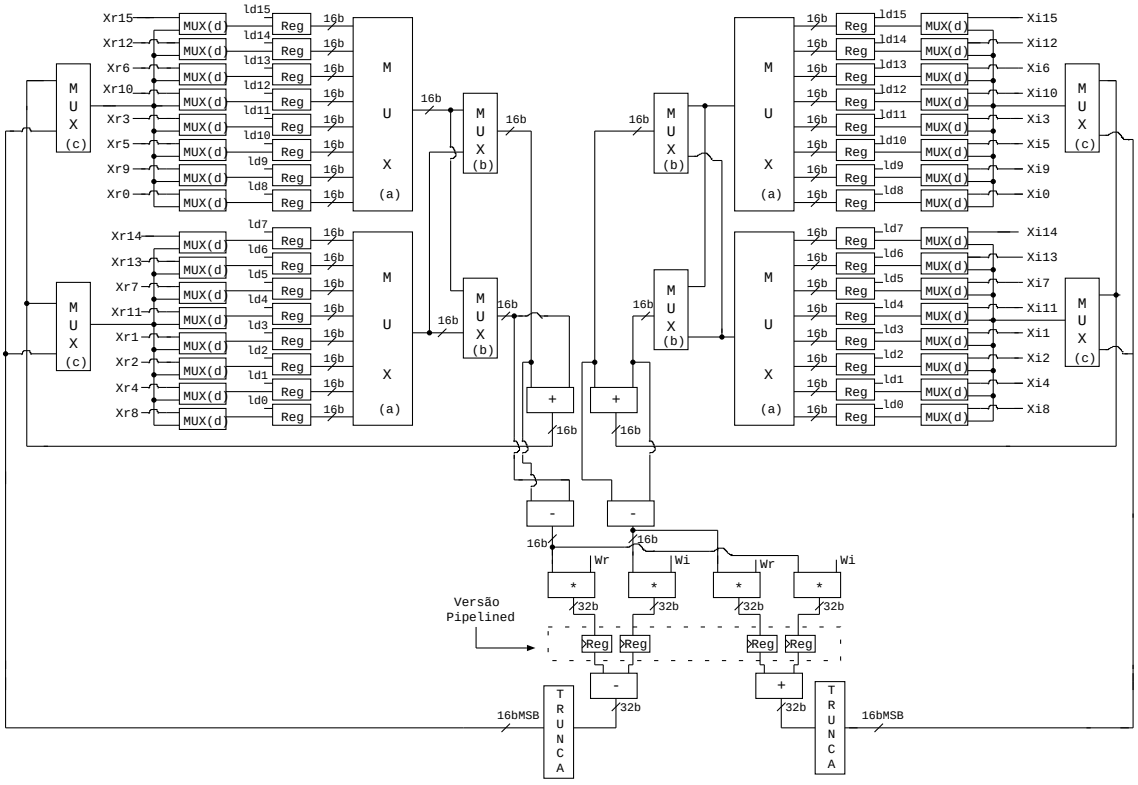

FIGURA 7.5 – Parte operativa das implementações da FFT seqüenciais.

Como foi observado anteriormente na Figura 5.15, o fluxo de dados do algoritmo FFT de fator comum com decimação em freqüência na base 2, utilizando 16

pontos, apresenta um total de 4 estágios, onde são utilizadas 8 borboletas em cada um dos estágios. Desta forma, o cálculo completo de uma transformada envolve 32 cálculos de termos reais e imaginários nas borboletas. Logo, para o processamento de uma transformada completa na arquitetura FFT seqüencial, mostrada na Figura 7.5 (16 pontos), são necessários 33 ciclos de relógio. Deste total, em 1 ciclo de relógio é efetuada a carga dos 16 pontos simultaneamente e nos demais 32 ciclos de relógio os dados são processados na borboleta.

Em uma arquitetura seqüencial, que utiliza apenas uma borboleta para processamento da transformada, os dados reais e imaginários são lidos de conjuntos de registradores. Os resultados de cálculo de cada estágio são armazenados nestes mesmos conjuntos, que podem ser vistos nos lados esquerdo e direito da Figura 7.5, para leitura e armazenamento dos dados reais e imaginários respectivamente. Como pode ser observado nesta mesma figura, a arquitetura seqüencial utiliza quatro grupos de circuitos multiplexadores (MUX(a), MUX(b), MUX(c), MUX(d)) em posições estratégicas do circuito. Um destes grupos (MUX(a)) é utilizado para disponibilizar à borboleta, os dados que são lidos dos registradores em diferentes instantes de tempo. O grupo MUX(b) é responsável por organizar os dados nas posições corretas das entradas da borboleta. Um outro grupo destes circuitos multiplexadores (MUX(c)) recebe os dados da borboleta e disponibiliza em suas saídas os dados para serem armazenados nas posições corretas dos registradores. Este grupo de multiplexadores recebe os dados de saída dos módulos denominados TRUNCA, mostrados na Figura 7.5. Estes módulos, compostos por *buffers*, apresentam nas suas saídas os 16 bits mais significativos dos resultados de cálculo efetuados pela borboleta. Estes 16 bits representam a parte mais significativa do valor fracionário entre -1 e 1. Os 16 bits mais significativos, após serem disponibilizados nas saídas do grupo de multiplexadores MUX(c), são habilitados no grupo de multiplexadores MUX(d), que também habilita as 16 entradas primárias simultaneamente a cada 33 ciclos de relógio. As 16 entradas primárias são posicionadas nas entradas do grupo de multiplexadores MUX(d), de tal forma que possibilite a habilitação de duas amostras diferentes a cada ciclo de relógio.

A parte de controle da arquitetura totalmente seqüencial (omitida na Figura 7.5) é responsável pela habilitação de 26 sinais de controle. Esta mesma quantidade de sinais é utilizada para habilitação dos circuitos das partes real e imaginária. Deste total, 16 sinais são utilizados para carregamento dos registradores, 6 sinais endereçam as saídas do grupo de multiplexadores MUX(a), 2 sinais são utilizados para endereçamento do grupo de multiplexadores MUX(c) e mais 2 sinais endereçam o grupo de multiplexadores MUX(d).

A utilização de quatro circuitos multiplicadores na estrutura da borboleta da arquitetura de uma FFT totalmente seqüencial gera uma quantidade considerável de *glitching* nas saídas destes circuitos. Desta forma, neste trabalho apresenta-se uma alternativa de implementação da estrutura seqüencial na forma *pipelined*, onde são utilizados registradores nas saídas dos circuitos multiplicadores, como mostrado nas linhas pontilhadas da Figura 7.5. Naturalmente, a introdução destes registradores aumenta a área da arquitetura FFT na forma *pipelined*, como mostrado na Tabela 7.9. Entretanto, apesar da maior área esta alternativa arquitetural consegue apresentar reduzido valor de atraso em relação à arquitetura na forma direta, como mostrado na Tabela 7.9. Isto ocorre pelo fato da introdução dos registradores na forma *pipelined* retirar do caminho crítico um circuito subtrator, no cálculo da

TABELA 7.9 – Valores área, período mínimo de relógio e potência normalizada por transformada para arquiteturas FFT seqüenciais.

Parâmetros	Direta	*Pipelined*	Dif.(%) *Pipe*→Direta
Área (Literais)	30327	32051	+5,68
Período Mínimo de Relógio (ns)	404,4	377,7	-6,6
Potência por transformada	195,5	141,6	-27,6

parte real, ou um circuito somador, no cálculo da parte imaginária, além dos circuitos de truncamento dos bits das saídas da borboleta. Ao fato de reduzir a profundidade lógica da arquitetura seqüencial, tem-se que a forma *pipelined* apresenta um menor valor de consumo de potência normalizada por transformada, como mostrado na Tabela 7.9. Isto devido à menor atividade de chaveamento, pela redução da atividade de *glitching* na saída dos circuitos multiplicadores. Os valores de potência por transformada da Tabela 7.9 foram obtidos usando n=33 e N=16 na Equação 7.10. A máquina de estados finita da parte de controle da arquitetura seqüencial foi gerada a partir da síntese de uma PLA. A potência consumida apresentada por esta máquina de estados representa aproximadamente 0,2% da potência total da arquitetura FFT seqüencial. Entretanto, embora este percentual seja reduzido em relação à potência total da arquitetura, deve-se observar que para arquiteturas FFT com maior número de pontos a quantidade de sinais de controle cresce de forma significativa e a geração da máquina de estados a partir da síntese de uma PLA se torna inviável. Isto pela complexidade lógica envolvida que pode fazer com que este circuito gere uma grande quantidade de *glitching*.

7.2.2 Arquiteturas Semi-Paralelas

Em uma arquitetura semi-paralela, os requisitos de *hardware* são duplicados e os dados podem ser processados na metade dos ciclos de relógio de uma arquitetura seqüencial. No algoritmo FFT, a duplicação do *hardware* implica a utilização de duas borboletas que fazem o processamento de quatro amostras reais e quatro amostras imaginárias simultaneamente. A Figura 7.6 mostra a parte operativa da arquitetura semi-paralela de uma FFT de 16 pontos nas formas direta e *pipelined*.

Da mesma forma como na arquitetura seqüencial, na arquitetura semi-paralela são utilizados grupos de registradores para leitura e armazenamento dos dados reais e imaginários. O fato de serem utilizadas duas borboletas para processamento dos dados faz aumentar a quantidade de circuitos multiplexadores, como mostrado na Figura 7.6. Estes circuitos multiplexadores têm as mesmas funções dos circuitos mostrados na arquitetura seqüencial, ou seja, são utilizados grupos de multiplexadores (MUX(a), MUX(b), MUX(c), MUX(d)) para habilitação dos dados a serem processados nas borboletas e habilitação dos resultados de processamento das transformadas, para armazenamento nos registradores. Todas as saídas dos multiplexadores do grupo MUX(a) são colocadas em todas as entradas dos multiplexadores do grupo MUX(b) para organização dos dados para as entradas das borboletas. Todas as saídas das borboletas são colocadas nas entradas dos multiplexadores dos grupos MUX(c), para que os dados sejam corretamente posicionados para armazena-

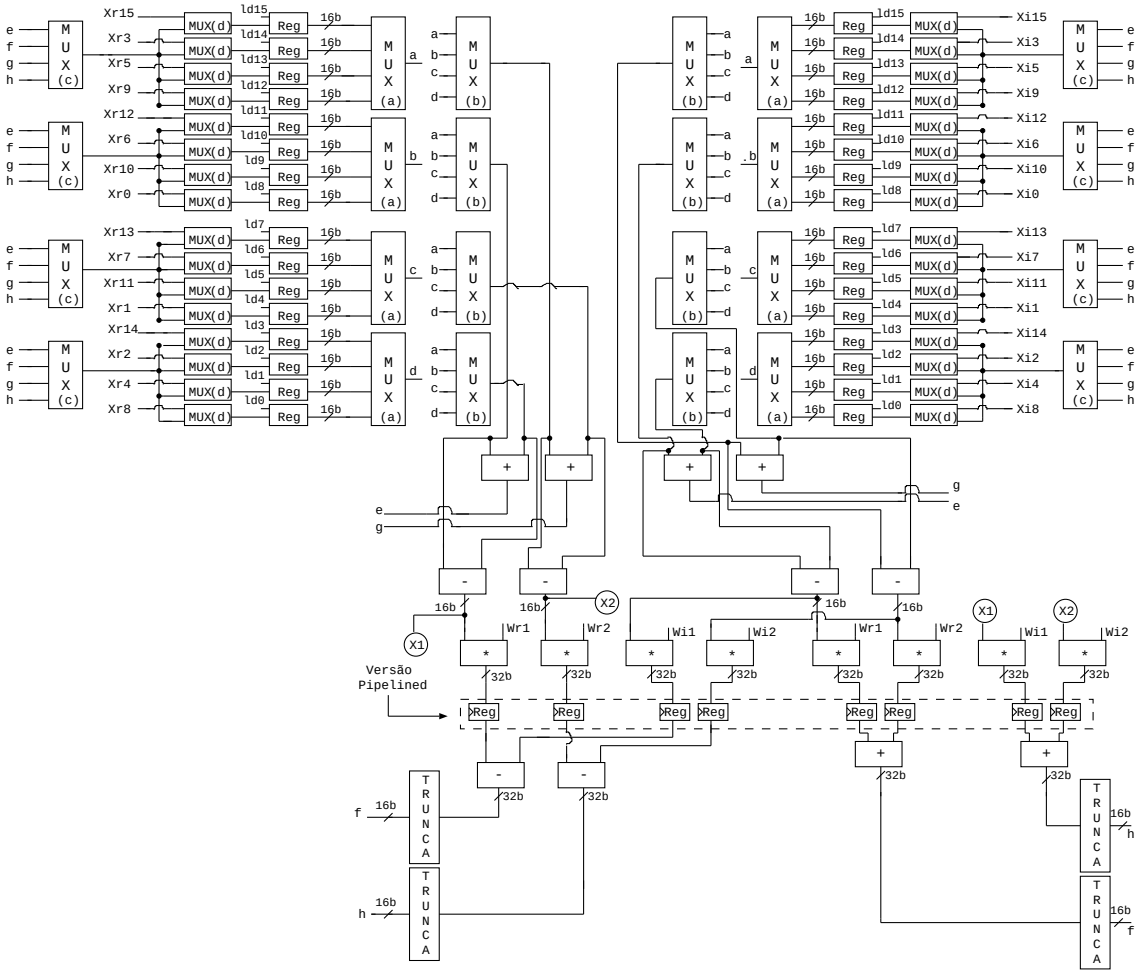

FIGURA 7.6 – Parte operativa das implementações da FFT semi-paralelas.

mento nos registradores. Estes dados são habilitados pelos multiplexadores do grupo MUX(d) que também habilitam as entradas primárias a cada 17 ciclos de relógio (16 ciclos para processamento dos dados e 1 ciclo para carga dos 16 pontos simultaneamente). Assim como nas arquiteturas seqüenciais, nas arquiteturas semi-paralelas as entradas primárias são posicionadas nas entradas do grupo de multiplexadores MUX(d), de tal forma que possibilite a habilitação de quatro amostras diferentes a cada ciclo de relógio. A parte de controle da arquitetura semi-paralela é composta por 40 sinais de controle, onde 16 sinais habilitam os registradores e os demais 24 sinais habilitam os circuitos multiplexadores.

O aspecto de se utilizar o dobro do *hardware* na arquitetura semi-paralela faz com que os coeficientes sejam divididos em dois grupos em cada estágio da FFT, o que leva à redução do tamanho da memória ROM para armazenamento dos coeficientes (omitida no desenho). O aumento do número de borboletas utilizado na arquitetura semi-paralela faz também aumentar a quantidade de operadores aritméticos na arquitetura FFT. Nesta arquitetura são utilizados 6 circuitos somadores, 6 circuitos subtratores e 8 circuitos multiplicadores. Particularmente, em relação aos circuitos multiplicadores, o aumento da sua utilização também torna mais intensa a quantidade de *glitching* gerada por estes circuitos. Desta forma, experimenta-se neste trabalho uma arquitetura semi-paralela na forma *pipelined*,

TABELA 7.10 – Valores área, período mínimo de relógio e potência normalizada por transformada para arquiteturas FFT semi-paralelas.

Parâmetros	Direta	*Pipelined*	Dif.(%) *Pipe*→Direta
Área (Literais)	50288	53246	+5,8
Período Mínimo de Relógio (ns)	403,2	381,7	-5,3
Potência por transformada	182,7	118,1	-35,4

TABELA 7.11 – Comparação de potência normalizada por transformada entre arquiteturas FFT seqüencial e semi-paralela com redução de V_{dd}.

Arquiteturas	Pot. p/ transf. Seqüencial	Pot. p/ transf. Semi-Paralela com $V_{dd}/2$	Dif(%) Semi-Par vs. Seq
Direta	195,5	182,7/4 = 45,7	-76,6
Pipelined	141,6	118,1/4 = 29,5	-79,2

com a introdução de registradores nas saídas dos circuitos multiplicadores, como mostrado nas linhas pontilhadas da Figura 7.6. Naturalmente, a introdução destes registradores faz aumentar a área da arquitetura *pipelined* em relação à forma direta, como mostrado na Tabela 7.10. Entretanto, embora apresente maior valor de área, a arquitetura na forma *pipelined* apresenta menor caminho crítico, o que possibilita a redução do período mínimo de relógio, como mostrado na Tabela 7.10. Como pode também ser observado nesta tabela, a arquitetura na versão *pipelined* apresenta menor valor de potência normalizada por transformada. Isto devido ao fato da significativa redução da atividade de *glitching* nas saídas dos circuitos multiplicadores. Os valores de potência por transformada da Tabela 7.10 foram obtidos considerando n=17 e N=16 na Equação 7.10. Da mesma forma, como realizado na arquitetura seqüencial, a máquina de estados finita da arquitetura semi-paralela é obtida a partir da síntese de uma PLA. Para a arquitetura semi-paralela, o consumo de potência desta máquina de estados representa uma parcela de aproximadamente 0,1% em relação à potência total da arquitetura. Embora esta parcela seja pequena em relação ao consumo global da arquitetura, vale ressaltar que para uma arquitetura FFT com um maior número de pontos esta estratégia não se torna atrativa devido à grande quantidade de *glitching* que é gerada pelo circuito combinacional obtido. Para a arquitetura semi-paralela, que apresenta uma maior quantidade de sinais de controle em relação à arquitetura totalmente seqüencial, este aspecto se torna mais crítico.

7.2.3 Comparações das Arquiteturas

Na seção anterior foi apresentada a arquitetura semi-paralela como uma alternativa para aumento da velocidade de processamento do algoritmo FFT de fator comum com decimação em freqüencia na base 2, a partir da redução do número de ciclos de relógio utilizados. Naturalmente, esta maior velocidade de processamento é obtida ao custo de aumento do *hardware*. Entretanto, apesar do aumento do *hard-*

ware é observado que as arquiteturas semi-paralelas nas formas direta e *pipelined* apresentam menores valores de potência normalizada por transformada em relação às arquiteturas totalmente seqüenciais. Este aspecto está relacionado ao fato dos circuitos multiplexadores serem mais simples nesta alternativa arquitetural, o que contribui para a redução da quantidade de níveis lógicos nestas arquiteturas. Em relação ao período mínimo de relógio, observa-se que as arquiteturas seqüencial e semi-paralela apresentam praticamente os mesmos valores, por apresentarem os mesmos caminhos críticos. Entretanto, a forma *pipelined* consegue reduzir o caminho crítico, e conseqüentemente o período mínimo de relógio, em ambas as alternativas arquiteturais.

Em relação à parte de controle das arquiteturas, observa-se que a máquina de estados das arquiteturas semi-paralelas apresenta um menor consumo de potência (0,542mW) em relação à máquina de estados das arquiteturas seqüenciais (0,681mW). De fato, embora as arquiteturas semi-paralelas apresentem um maior número de sinais de controle, observa-se que estas arquiteturas operam na metade dos ciclos de relógio, tendo-se desta forma, uma máquina de controle mais simples com menos estados, e conseqüentemente, com menor consumo de potência.

A forma *pipelined* apresenta-se como uma boa alternativa para redução de potência nas arquiteturas FFT seqüencial e semi-paralela. Este fato está relacionado com a redução da quantidade de *glitching* nas saídas dos circuitos multiplicadores. Em particular, este aspecto é mais representativo na arquitetura semi-paralela que utiliza o dobro de circuitos multiplicadores utilizados pela arquitetura seqüencial. Além disto, em relação às arquitetura semi-paralelas, observa-se que os seus valores de consumo de potência podem ser ainda mais reduzidos, pela possibilidade de redução da tensão de alimentação pela metade. Isto pelo fato destas arquiteturas operarem com a metade de ciclos de relógio utilizadas em uma arquitetura totalmente seqüencial. Este resultado é mostrado na Tabela 7.11.

Desta forma, observa-se que quando o primeiro aspecto de projeto a ser considerado no algoritmo FFT é a redução do consumo de potência, a arquitetura semi-paralela se torna atrativa. Em particular, a alternativa *pipelined* consegue uma maior redução de potência, com redução do atraso e acréscimo em latência de apenas 1 ciclo de relógio. Naturalmente, esta maior redução de potência nas arquiteturas semi-paralelas apresenta uma penalidade em área com o dobro do *hardware* apresentado por uma arquitetura totalmente seqüencial. Desta forma, se a área é um dos principais parâmetros a ser considerado no projeto da estrutura FFT, a arquitetura seqüencial deve ser a primeira escolha. Neste caso, a escolha da forma *pipelined* nesta arquitetura se torna atrativa devido à possibilidade de redução dos parâmetros de período mínimo de relógio e consumo de potência, com baixa penalidade de área.

7.2.4 Arquiteturas com Multiplicadores na Base 2^m

Um dos principais aspectos das arquiteturas FFT mostradas anteriormente é a grande quantidade de circuitos multiplicadores utilizados na estrutura da borboleta. Estes multiplicadores são responsáveis pela maior parte do consumo de potência das arquiteturas FFT. Nesta parte do trabalho serão mostrados os principais resultados de área, atraso e potência provenientes da aplicação de multiplicadores na base 2^m. Os resultados serão comparados com a arquitetura utilizando valor m=1.

Como demonstrado na seção anterior, as arquiteturas na forma *pipelined*, embora apresentem maior valor de área, apresentam menores valores de atraso e con-

sumo de potência devido à utilização de registradores nas saídas dos multiplicadores, que além de reduzir o caminho crítico das arquiteturas consegue também reduzir a atividade de *glitching*. Desta forma, nesta seção serão apresentados os resultados de área, atraso e consumo de potência das arquiteturas na forma *pipelined*.

7.2.4.1 Resultados de Área, Período Mínimo de Relógio e Potência por Transformada

Como as arquiteturas FFT seqüencial e semi-paralela apresentam uma grande quantidade de circuitos multiplicadores nas borboletas, os valores dos parâmetros de área, período mínimo de relógio e consumo de potência normalizada por transformada estão associados às características dos circuitos multiplicadores. Como pode ser observado na Tabela 7.12, as arquiteturas com multiplicadores utilizando valores de m maiores do que 1 apresentam os maiores valores de área. Isto devido ao maior número de portas lógicas apresentado nos módulos básicos dos circuitos multiplicadores com valores de m maiores do que 1, como mostrado no Capítulo 6. Particularmente, as arquiteturas FFT com operadores aritméticos com valor de m=4, onde a complexidade dos módulos básicos dos multiplicadores é mais elevada, apresentam os maiores valores de área.

TABELA 7.12 – Valores de área para alternativas arquiteturais e diferentes valores de m na forma *pipelined*.

Alternativas Arquiteturais	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Seqüencial	m=1	32051	-	-	-
	m=2	33683	+5,1	36499	+13,8
	m=4	57163	+78,3	104447	+225,8
Semi-Paralela	m=1	53246	-	-	-
	m=2	56510	+6,1	62142	+16,7
	m=4	103470	+94,3	198038	+271,9

Embora as arquiteturas FFT com valores de m=2 e 4 apresentem maiores valores de área, estas arquiteturas podem apresentar menores valores de atraso em relação às arquiteturas com operadores Binários com m=1, como pode ser visto na Tabela 7.13. Como pode ser observado nesta mesma tabela, as arquiteturas seqüencial e semi-paralela apresentam praticamente os mesmos valores de atraso, pois estas arquiteturas apresentam o mesmo caminho crítico. As arquiteturas com operadores em código Híbrido apresentam os maiores valores de atraso. Este aspecto está associado com o maior atraso apresentado pelos multiplicadores Híbridos, como mostrado no Capítulo 6. Entretanto, este valor de atraso se torna bem maior para os operadores Híbrido com m=4, pois além do fato dos multiplicadores com este valor de m apresentarem os maiores valores de atraso, a conversão entre os códigos Binário e Híbrido nos barramentos de dados, com este valor de m, exige a utilização de um maior número de portas lógicas EXOR. Outro aspecto a ser considerado é que embora os multiplicadores Binário e Híbrido com m=4 apresentem menores valores de atraso em relação aos multiplicadores com m=2, observa-se que as arquiteturas FFT com estes operadores apresentam maiores valores de atraso. Isto ocorre devido ao fato da

soma dos atrasos dos caminhos críticos nas arquiteturas com estes operadores apresentarem maiores valores. Embora os multiplicadores Híbridos com valores de m=2 e 4 apresentem menores valores de atraso em relação aos multiplicadores convencionais m=1, observa-se que as arquiteturas FFT com estes operadores apresentam os maiores valores de atraso. Este aspecto está associado com o grande número de portas lógicas EXOR utilizadas na estrutura da borboleta, para conversão entre os códigos Binário e Híbrido.

TABELA 7.13 – Valores de atraso em ns para alternativas arquiteturais e diferentes valores de m na forma *pipelined*.

Alternativas Arquiteturais	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Seqüencial	m=1	377,7	-	-	-
	m=2	332,2	-12,0	413,2	+9,4
	m=4	372,9	-1,3	474,3	+25,5
Semi-Paralela	m=1	383,7	-	-	-
	m=2	338,2	-11,8	420,2	+9,5
	m=4	378,9	-1,2	480,3	+25,2

Embora as arquiteturas com operadores em código Híbrido apresentem expressivos maiores valores de atraso em relação às arquiteturas com operadores Binários, observa-se que a diferença em termos de consumo de potência não se torna tão significativa, como pode ser observado na Tabela 7.14. Este fato está associado com a maior correlação nas entradas dos circuitos multiplicadores com os sinais em código Híbrido. Além disto, a maior correlação nos valores dos coeficientes em código Híbrido contribui para que haja uma redução significativa do consumo de potência com estes operadores. Esta maior correlação nos coeficientes é mais significativa nas arquiteturas com operadores em código Híbrido com m=4, pois embora os multiplicadores Híbridos com m=4 apresentem maior valor de consumo de potência, como mostrado no Capítulo 5, as arquiteturas FFT com estes operadores apresentam menores valores de consumo de potência em relação às arquiteturas com operadores em código Híbrido com m=2. Outro aspecto a ser destacado na Tabela 7.14 é o menor consumo de potência nas arquiteturas semi-paralelas em relação às arquiteturas seqüenciais. Isto se deve à estrutura mais simples dos circuitos multiplexadores e ao aspecto da maior correlação que se faz presente em maior intensidade quando se utiliza uma maior quantidade de circuitos multiplicadores. Como também pode ser observado na Tabela 7.14, o menor consumo de potência observado nos multiplicadores Binários com m=2 e 4 faz com que as arquiteturas seqüenciais e semi-paralelas com estes operadores apresentem menores valores de consumo de potência normalizada por transformada. Particularmente, nas arquiteturas com operadores Binários com m=4, esta redução de potência normalizada por transformada é mais significativa.

TABELA 7.14 – Valores de potência normalizada por transformada para alternativas arquiteturais e diferentes valores de m na forma *pipelined*.

Alternativas Arquiteturais	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Seqüencial	m=1	141,6	-	-	-
	m=2	96,0	-31.5	110,4	-21,7
	m=4	84,8	-40,0	100,8	-28,6
Semi-Paralela	m=1	118,1	-	-	-
	m=2	81,6	-30,5	92,8	-20,6
	m=4	72,0	-38,3	88,0	-25,7

7.2.5 Comparação das Arquiteturas com Multiplicador Booth Modificado

Como abordado no Capítulo 6, o multiplicador Booth Modificado apresenta uma maior quantidade de interconexões e maior valor de atraso comparado às arquiteturas de multiplicador *array* propostas neste trabalho. Estes aspectos contribuem para que o multiplicador Booth apresente maior valor de consumo de potência em relação aos multiplicadores *array*, como apresentado anteriormente na Tabela 6.7. Desta forma, nas arquiteturas FFT seqüencial e semi-paralela, que utilizam uma grande quantidade de circuitos multiplicadores, a utilização do multiplicador Booth faz com que estas arquiteturas, na forma direta, apresentem um elevado consumo de potência em relação às arquiteturas com multiplicadores *array* com m=2, como mostrado na Tabela 7.15.

TABELA 7.15 – Valores de potência normalizada por transformada para arquiteturas seqüencial e semi-paralela na forma direta com multiplicadores na base 4 (m=2).

Alternativas Arquiteturais	Booth	Binário	Diferença (%) Bin vs. Booth	Híbrido	Diferença (%) Hib vs. Booth
Seqüencial	262,2	139,8	-46,6	152,4	-41,8
Semi-Paralela	244,9	130,4	-46,7	149,3	-39,0

O elevado número de circuitos multiplicadores utilizado na borboleta da FFT produz uma grande quantidade de *glitching*, o que eleva a potência com multiplicador Booth. Como abordado na seção anterior, a forma *pipelined* faz com que as arquiteturas FFT reduzam os seus consumos de potência pelo fato da introdução de registradores nas saídas dos circuitos multiplicadores reduzir a atividade *glitching* ao longo dos circuitos. Desta forma, para as arquiteturas FFT seqüencial e semi-paralela que utilizam o multiplicador Booth, este tipo de alternativa arquitetural apresenta uma grande redução de consumo de potência quando comparado à forma direta, como pode ser visto nas Tabelas 7.15 e 7.16.

Apesar das arquiteturas com multiplicador Booth na forma *pipelined* apresentarem uma grande redução de potência normalizada por transformada em relação às arquiteturas na forma direta, pela redução da atividade de *glitching* nas saídas

TABELA 7.16 – Valores de potência normalizada por transformada para arquiteturas seqüencial e semi-paralela na forma *pipelined* com multiplicadores na base 4 (m=2).

Alternativas Arquiteturais	Booth	Binário	Diferença (%) Bin vs. Booth	Híbrido	Diferença (%) Hib vs. Booth
Seqüencial	156,6	96,0	-38,7	110,4	-29,5
Semi-Paralela	144,8	81,6	-43,6	92,8	-35,9

dos multiplicadores, observa-se na Tabela 7.16 que as arquiteturas com multiplicadores *array* apresentam menores consumos de potência, também nesta alternativa arquitetural. De fato, este aspecto está relacionado com a estrutura mais simples e menor profundidade lógica dos multiplicadores *array*, como apresentado no Capítulo 6. Além disto, as arquiteturas com multiplicadores *array* também apresentam uma considerável redução da atividade de *glitching* nas saídas dos circuitos multiplicadores, que também leva a uma redução de consumo de potência normalizada por amostra nas arquiteutras com estes operadores, como pode ser comparado nas Tabelas 7.15 e 7.16.

7.2.6 Arquitetura FFT com Maior Número de Pontos

Em aplicações da área DSP, tais como modems e terminais móveis sem fio, é comum a utilização de arquiteturas FFT com maior número de pontos [HE 98, LI 99]. Neste trabalho, apresentam-se arquiteturas FFT de fator comum na base 2 com decimação em freqüência nas formas direta e *pipelined* para diferentes números de amostras. Estas arquiteturas são produzidas a partir de um algoritmo em linguagem de programação C, que permite a produção de arquiteturas FFT totalmente seqüenciais no formato BLIF. Nesta parte do trabalho serão apresentadas as características e resultados de consumo de potência de arquiteturas totalmente seqüenciais de 64 pontos nas formas direta e *pipelined*. Embora o algoritmo proposto possa produzir arquiteturas FFT totalmente seqüenciais nas formas direta e *pipelined* para diferentes números de amostras, não foi possível obter resultados de consumo de potência para arquiteturas com valores de amostras acima de 64. Isto devido à limitação da ferramenta de simulação de potência utilizada neste trabalho (SLS).

7.2.6.1 Características da Arquitetura

O fluxo de dados de uma arquitetura FFT de fator comum na base 2 com decimação em freqüência utilizando 64 pontos apresenta 6 ($log_2 64$) estágios. Em cada um destes estágios são utilizadas 32 borboletas (64/2). O processamento completo da FFT de 64 pontos envolve 192 ($=32 \times 6$) cálculos de valores reais e imaginários. Logo, para o processamento da transformada completa em uma arquitetura totalmente seqüencial, são necessários 193 ciclos de relógio. Deste total, utiliza-se 1 ciclo de relógio para o carregamento dos 64 pontos simultaneamente e nos demais 192 ciclos são efetuados os cálculos das transformadas na borboleta. No cálculo das transformadas são utilizados 192 coeficientes reais e 192 coeficientes imaginários, que ficam armazenados em memória ROM.

A arquitetura FFT totalmente seqüencial de 64 pontos apresenta a mesma

estrutura mostrada na Figura 7.5, para uma FFT de 16 pontos, utilizando a mesma borboleta apresentada nesta figura. Entretanto, pelo fato da arquitetura utilizar um maior número de pontos, esta apresenta uma maior quantidade de registradores (128) para leitura dos dados e armazenamento dos resultados. Os multiplexadores utilizados na arquitetura FFT de 64 pontos estão divididos nos mesmos grupos mostrados na Figura 7.5, representados por MUX(a), MUX(b), MUX(c) e MUX(d). No entanto, para a arquitetura de 64 pontos, verifica-se o aumento da quantidade de multiplexadores do grupo MUX(d), que habilitam as entradas primárias e do grupo MUX(a), que agora apresentam um maior número de entradas (64).

Com o aumento da quantidade de registradores e multiplexadores, a arquitetura seqüencial de 64 pontos apresenta um maior número de sinais de controle para habilitação destes circuitos, tendo-se um total de 78 sinais. Deste total, utilizam-se 64 sinais para habilitação dos registradores e 10 sinais para endereçamento dos multiplexadores do grupo MUX(a) (5 sinais para os multiplexadores do grupo MUX(a) superior e 5 sinais para os multiplexadores do grupo MUX(a) inferior). Os restantes 4 sinais são utilizados para endereçamento dos multiplexadores dos grupos MUX(b) e MUX(c).

Para a redução da atividade de *glitching* nas saídas dos circuitos multiplicadores utilizados na borboleta, apresenta-se uma arquitetura totalmente seqüencial de 64 pontos na forma *pipelined*. A arquitetura utiliza registradores nas saídas dos circuitos multiplicadores, da mesma forma como mostrado nas linhas pontilhadas da Figura 7.5.

A seguir serão apresentados resultados de consumo de potência normalizada por transformada das arquiteturas FFT de 64 pontos nas formas direta e *pipelined*. Estes resultados foram obtidos usando n=193 e N=64 na Equação 7.10.

7.2.6.2 Aplicação de Multiplicadores *Array* na Base 2^m

Da mesma forma como abordado na seção anterior para arquiteturas FFT de 16 pontos, observa-se que a utilização dos multiplicadores *array* na base 2^m nas arquiteturas FFT totalmente seqüenciais de 64 pontos podem apresentar diferentes valores de consumo de potência, devido às diferentes características dos operadores aritméticos com diferentes valores de m. A Tabela 7.17 apresenta os valores de potência normalizada por transformada na arquitetura FFT seqüencial na forma direta com multiplicadores *array* utilizando diferentes valores de m. Os valores percentuais apresentados nesta tabela, representam comparação com a arquitetura FFT utilizando multiplicador Binário com m=1.

TABELA 7.17 – Valores de potência normalizada por transformada para arquiteturas totalmente seqüenciais na forma direta com diferentes valores de m.

Alternativa Arquitetural	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Totalmente Seqüencial	m=1	5772,8	-	-	-
	m=2	3904,0	-32,4	4627,2	-19,8
	m=4	3616,0	-37,4	4569,6	-20,8

A maior complexidade da arquitetura FFT de 64 pontos, com uma maior

quantidade de circuitos registradores e multiplexadores, além da maior atividade de transição nos barramentos de dados, faz com que as arquiteturas com operadores aritméticos Binário e Híbrido com diferentes valores de m, apresentem um elevado consumo de potência normalizada por tranformada, como mostrado na Tabela 7.17. Como pode ser observado nesta tabela, a utilização de multiplicadores *array* com valores de m=2 e 4 em códigos Binário e Híbrido reduz o consumo de potência normalizada por transformada em relação à arquitetura com m=1. Este resultado segue a mesma tendência apresentada na seção anterior para as arquiteturas FFT seqüenciais com 16 pontos. Isto devido ao fato de ambas as arquiteturas utilizarem a mesma borboleta para processamento das transformadas.

Como visto anteriormente, a maior atividade de transição nos barramentos de dados faz com que a arquitetura FFT de 64 pontos apresente um elevado consumo de potência. Desta forma, a utilização da versão *pipelined*, com a redução da atividade de *glitching* nas saídas dos circuito multiplicadores faz com que as arquiteturas com operadores aritméticos Binário e Híbrido, com diferentes valores de m, apresentem grandes reduções de potência por transformada, como mostrado na Tabela 7.18. O grande impacto em redução de potência das arquiteturas na forma *pipelined* pode ser comparado nas Tabelas 7.17 e 7.18, onde se observa reduções do consumo de potência por transformada destas arquiteturas em até mais de 40% em relação às arquiteturas na forma direta.

TABELA 7.18 – Valores de potência normalizada por transformada para arquiteturas totalmente seqüenciais na forma *pipelined* com diferentes valores de m.

Alternativa Arquitetural	Grupo de Bits	Operadores			
		Binário	(%)	Híbrido	(%)
Totalmente Seqüencial	m=1	3737,6	-	-	-
	m=2	2419,2	-35,3	2886,4	-22,7
	m=4	2123,5	-43,2	2953,6	-20,9

7.2.6.3 Comparação com Arquitetura Utilizando Multiplicador Booth

Para arquiteturas FFT de 64 pontos totalmente seqüenciais utilizando um multiplicador Booth Modificado, o aspecto da atividade de *glitching* se torna mais crítico, visto que estes multiplicadores apresentam uma grande propagação destes sinais ao longo da sua estrutura, como visto anteriormente no Capítulo 6. Este maior consumo de potência normalizada por transformada da arquitetura com este multiplicador pode ser visto na Tabela 7.19, que apresenta resultados de comparação com as arquiteturas utilizando multiplicadores *array* Binário e Híbrido na base 4 (m=2) na forma direta.

Como pode ser visto na Tabela 7.19, na forma direta a arquitetura com multiplicador Booth apresenta um elevado consumo de potência em relação às arquiteturas com multiplicadores Binário e Híbrido. No entanto, da mesma forma como apresenta um grande impacto na redução de potência com operadores *array* com diferentes valores de m, a arquitetura FFT seqüencial de 64 pontos na forma *pipelined* também apresenta uma significativa redução do seu consumo de potência normalizada por transformada com a utilização do multiplicador Booth, como pode ser

TABELA 7.19 – Comparações de valores de potência normalizada por transformada para arquiteturas totalmente seqüenciais na forma direta com multiplicadores *array* e Booth na base 4 (m=2).

Alternativa Arquitetural	Booth	Binário	Diferença (%) Bin vs. Booth	Híbrido	Diferença (%) Hib vs. Booth
Seqüencial	6365,4	3904	-38,7	4627,2	-27,3

visto na Tabela 7.20. Como pode ser comparado nas Tabelas 7.19 e 7.20, a arquitetura na forma *pipelined* com multiplicador Booth apresenta uma redução de aproximadamente 40% no seu consumo de potência normalizada por transformada em relação à arquitetura na forma direta, da mesma forma como apresentado para arquiteturas com multiplicadores *array* na base 2^m. Assim sendo, embora a redução de potência da arquitetura na forma *pipelined* com multiplicador Booth seja significativa, as arquiteturas com multiplicadores Binário e Híbrido ainda apresentam significativamente menor valor de consumo de potência, pois as arquiteturas com estes operadores também apresentam um grande impacto em redução de potência.

TABELA 7.20 – Comparações de valores de potência normalizada por transformada para arquiteturas totalmente seqüenciais na forma *pipelined* com multiplicadores *array* e Booth na base 4 (m=2).

Alternativa Arquitetural	Booth	Binário	Diferença (%) Bin vs. Booth	Híbrido	Diferença (%) Hib vs. Booth
Seqüencial	3786,2	2419,2	-36,1	2886,4	-23,7

7.3 Resumo

Neste capítulo foram abordadas implementações de arquiteturas dedicadas para os algoritmos de filtro FIR e FFT. Foram efetuadas explorações arquiteturais que possibilitaram implementações de arquiteturas nas alternativas totalmente paralela, totalmente seqüencial e semi-paralela para o filtro FIR. No algoritmo FFT, a implementação de uma alternativa totalmente paralela se torna inviável devido ao grande número de operadores aritméticos envolvido. Desta forma, para este algoritmo foram implementadas arquiteturas nas formas totalmente seqüencial e semi-paralela. Para o filtro FIR, as alternativas arquiteturais foram implementadas nas formas direta, *pipelined* e transposta. Para o algoritmo FFT foram apresentadas alternativas arquiteturais nas formas direta e *pipelined*. Para ambos os algoritmos foram levados em consideração aspectos de área, velocidade de operação e consumo de potência. Para as implementações do filtro FIR, a alternativa totalmente paralela apresentou os melhores resultados de potência normalizada por amostra. Isto devido à possibilidade de utilização nestas arquiteturas de multiplicadores desloca-soma que apresentam uma menor atividade de transição. Para esta alternativa arquitetural, a maior redução de potência por amostra foi obtida ao custo de uma maior área. Para as arquiteturas seqüencial e semi-paralela, não é possível a utilização deste

tipo de operador devido à reutilização do *hardware* tanto quanto possível tornando a operação dos circuitos associativa e comutativa. Para estas arquiteturas, tanto no filtro FIR como no algoritmo FFT, foi utilizado um circuito multiplicador tipo *array* em complemento de 2. Como foi observado nestes algoritmos, as arquiteturas semi-paralelas apresentam menores valores de consumo de potência. Isto devido ao fato de serem utilizados circuitos multiplexadores mais simples nas arquiteturas, além da maior correlação dos dados, quando se utiliza um maior número de circuitos multiplicadores. Este aspecto é mais significativo no algoritmo FFT, onde a estrutura da borboleta utiliza um maior número de circuitos multiplicadores. A utilização dos multiplicadores *array* nas arquiteturas dos algoritmos de filtro FIR e FFT faz com que estas arquiteturas consumam menos potência em relação às arquiteturas com multiplicador Booth. No algoritmo FFT, onde uma maior quantidade de circuitos multiplicadores é utilizada, esta diferença atingiu valores próximos dos 50%. Este valor elevado de ganho está relacionado com a estrutura mais simples dos multiplicadores *array*. Além disto, o multiplicador Booth produz uma grande quantidade de *glitching*, o que eleva significativamente o consumo de potência das arquiteturas com este multiplicador. Este aspecto foi comprovado com a apresentação de resultados de consumo de potência nas arquiteturas na forma *pipelined*. Nestas arquiteturas, verificou-se a grande redução de potência nas arquiteturas com multiplicador Booth, devido à introdução de registradores nas saídas dos circuitos multiplicadores, que reduzem a quantidade de *glitching* produzida por estes circuitos. Entretanto, como foi observado, mesmo nesta alternativa arquitetural, as arquiteturas de filtro FIR e FFT com multiplicadores *array* ainda apresentam menores valores de potência, pois as arquiteturas na forma *pipelined* com estes operadores aritméticos também apresentam redução da atividade de *glitching*, e conseqüentemente, do consumo de potência. Este aspecto também foi comprovado em arquiteturas FFT com 64 pontos, onde foi visto que a arquitetura na forma *pipelined* pode apresentar um grande impacto na redução da atividade de *glitching* nas saídas dos circuitos multiplicadores, com reduções de consumo de potência por transformada em até 40% em relação às arquiteturas na forma direta. Como também foi observado neste capítulo, as arquiteturas com operadores em código Binário apresentam menores valores de consumo de potência normalizada por amostra nas arquiteturas de filtro FIR e menores valores de potência normalizada por transformada nas arquiteturas FFT. Entretanto, a diferença para as arquiteturas com operadores em código Híbrido não é signifcativa.

8 Conclusão

A crescente demanda por equipamentos portáteis e equipamentos móveis bem como o fato de não haver nos últimos anos um incremento significativo na eficiência da tecnologia de baterias, tem motivado o estudo efetivo de técnicas visando a redução de potência de circuitos integrados.

Um ponto importante a ser destacado na estimativa de potência é o padrão do sinal de entrada dos circuitos. No nível elétrico por exemplo, a dificuldade de estimativa de potência cresce com a complexidade e o número de entradas dos circuitos. Para circuitos, como multiplicadores e somadores de n entradas, por exemplo, pode-se ter 2^{2^n} seqüências de padrões de entrada de sinais. No alto nível de abstração, torna-se importante a utilização de sinais conhecidos nas entradas dos operadores aritméticos com alto coeficiente de correlação [LAN 95, RAM 97, COS 99].

Neste trabalho, foi utilizado um sinal senoidal nas entradas dos circuitos como parâmetro de estabelecimento das suas atividades de transição. Desta forma, diferentes técnicas para redução da atividade de transição em operadores aritméticos foram investigadas com base na correlação existente neste sinal e na sua propagação ao longo dos circuitos. Igualmente foi feita a verificação da redução de potência consumida quando os operadores recebem entradas pseudo-aleatórias (distribuição uniforme).

A redução do número de transições no sentido de minimizar o consumo de potência é o principal objetivo de técnicas que usam diferentes esquemas de codificação em barramentos. Neste trabalho foram implementados operadores aritméticos que operam diretamente com diferentes códigos de entrada. Estes operadores foram comparados com operadores que operam com código Binário. Foi apresentado um código Híbrido como um caminho alternativo para implementação de operadores aritméticos com ênfase a estruturas de multiplicadores do tipo *array*. Foram investigadas comparações de desempenho de arquiteturas de multiplicadores *array* em códigos Binário e Híbrido e os resultados mostraram que apesar dos multiplicadores Híbridos apresentarem maiores valores de área, estas arquiteturas podem apresentar valores de atraso e consumo de potência próximos dos operadores Binários. Estes resultados mostram que existe outra possibilidade de implementação de operadores em diferente código do Binário, com possibilidades de redução de atraso e consumo de potência. Embora não tenha sido o principal enfoque deste trabalho, acreditamos que para barramentos com elevados valores de capacitância de carga, os operadores Híbridos podem ser uma boa alternativa para redução de potência. Este aspecto foi introduzido no Capítulo 5, onde foi mostrada a utilização de *buffers* nas entradas dos circuitos multiplicadores representando valores de capacitâncias de carga. Como foi observado, a introdução de um número elevado destes circuitos nas entradas dos operadores em código Híbrido revelou menores valores de consumo de potência em relação aos operadores Binários. Entretanto, mesmo mostrando esta tendência, acreditamos que para uma análise mais detalhada desta influência, faz-se necessário a implementação dos circuitos no nível físico, onde uma estimativa mais precisa do consumo de potência nos operadores com diferentes valores de capacitância de carga possa ser obtida.

Neste trabalho foi proposta uma nova arquitetura de multiplicador *array* na base 2^m em complemento de 2 em códigos Binário e Híbrido. Estas arquiteturas

apresentam a mesma regularidade de um multiplicador *array* convencional, tendo entretanto, um menor número de linhas de produtos parciais. Foram apresentados resultados de arquiteturas com valores de m=2 e 4. Para m maior do que 4, estas arquiteturas não se mostraram eficientes em reduções de atraso e consumo de potência devido à grande complexidade dos módulos básicos dos produtos. Entretanto, como foi observado, as arquiteturas com valores de m=2 e 4 se mostram mais eficientes em termos de atraso e consumo de potência em relação à arquitetura Binário com m=1. Isto devido à redução das linhas de produtos parciais nestes operadores. Os multiplicadores Binário e Híbrido com m=2 foram comparados à arquitetura de um multiplicador Booth Modificado, que representa o estado da arte em redução do consumo de potência. Os resultados mostraram que apesar dos maiores valores de área, a maior regularidade e a estrutura mais simples dos multiplicadores *array* proporcionam uma significativa redução de potência e uma pequena redução de atraso em relação ao multiplicador Booth.

A aplicação dos operadores aritméticos na base 2^m em circuitos da área de processamento digital de sinais foi também investigado. Em particular, foram utilizados os algoritmos de filtro FIR e FFT para exploração arquitetural e aplicação de técnicas de baixa potência. Foram implementadas alternativas arquiteturais destes algoritmos. Com a aplicação destes operadores, foi mostrado que a forma *pipelined* apresenta melhor desempenho e maior redução de potência entre as arquiteturas implementadas. Isto devido à introdução de registradores nas saídas dos circuitos multiplicadores que reduz o caminho crítico e a atividade de *glitching* nas arquiteturas. Entre as alternativas arquiteturais implementadas no algoritmo de filtro FIR, foi mostrado que a alternativa totalmente paralela se mostrou mais eficiente em termos de consumo de potência por amostra. Isto devido à possibilidade de utilização de circuitos multiplicadores tipo desloca-soma em suas arquiteturas. Entretanto, a maior eficiência em consumo de potência por amostra neste tipo de alternativa arquitetural é obtida ao custo de uma maior área.

Nas alternativas arquiteturais seqüencial e semi-paralela não é possível a utilização de circuitos multiplicadores tipo desloca-soma. Entretanto, observa-se que nestas alternativas arquiteturais é possível a manipulação dos coeficientes no sentido de redução da atividade de chaveamento nas entradas dos circuitos multiplicadores. Isto devido ao tipo de operação comutativa e associativa presentes nestas alternativas arquiteturais. Os resultados de aplicação da técnica de ordenação de coeficientes mostraram que é possível uma redução de consumo de potência nas alternativas arquiteturais sequencial e semi-paralela. Foi mostrado que a técnica é mais efetiva nas arquiteturas FFT, onde o grau de oportunidade de aplicação da técnica em uma maior quantidade de grupos de coeficientes permite uma maior redução da distância de *Hamming* em uma maior quantidade de coeficientes. Outro aspecto a ser destacado para a maior eficiência da técnica de ordenação de coeficientes no algoritmo FFT é a maior correlação entre as palavras, pela característica de proximidade dos valores dos coeficientes. Em particular, foi mostrado que a maior correlação entre os coeficientes pode favorecer de uma forma mais efetiva as arquiteturas que utilizam operadores aritméticos em código Híbrido.

8.1 Principais contribuições

As contribuições relevantes deste trabalho podem ser divididas em:

- Implementação de diferentes operadores aritméticos usando códigos *Transition*, Gray, Redundante e Híbrido. Dentre estes, o que se mostrou mais atrativo foi o código Híbrido. O código Híbrido representa o compromisso entre a mínima dependência das entradas apresentada pelo código Binário e a característica de baixa atividade de chaveamento apresentada pelo código Gray. Neste trabalho foi apresentada uma arquitetura *array* na base 2^m, operando neste código. Os resultados mostraram que esta arquitetura apresenta valores de atraso e consumo de potência próximos à arquitetura com código Binário e melhores valores destes parâmetros comparados com o multiplicador Booth. Para valor maior de capacitância de barramento, operadores em código Híbrido podem ser bastante mais eficientes em termos de consumo de potência. Estes resultados mostram uma outra possibilidade de operadores aritméticos, operando em um código diferente do Binário;

- Implementação de operadores aritméticos com maiores valores de m. Na etapa inicial deste trabalho o enfoque foi a aplicação de operadores em código Híbrido com grupos de bits de tamanho $m = 2$ sem sinal. Entretanto, como pôde ser observado, é possível se ter operadores aritméticos em códigos Híbrido e Binário com maiores valores de m. Neste trabalho, foram implementados operadores sem sinal com grupos de bits de tamanhos m=2, 4 e 8. Neste caso, foi observado que para valores de m maiores do que 4, a complexidade dos módulos aritméticos aumenta consideravelmente inviabilizando a aplicação para baixa potência. Neste particular, as arquiteturas com valores de m=2 e 4 apresentaram reduções de atraso e potência comparado com o multiplicador *array* convencional com m=1;

- Implementação de operadores aritméticos em complemento de 2. Na área DSP uma das principais operações realizadas envolve a multiplicação de dados com sinal. Em operações de multiplicação com sinal para baixa potência, o algoritmo mais utilizado é o Booth Modificado que opera em números representados em complemento de 2. Este algoritmo apresenta como principal característica a redução do número de linhas dos produtos parciais que reduz a quantidade de operadores aritméticos e conseqüentemente a dissipação de potência. No nosso trabalho, apresentamos uma nova abordagem para a produção de operandos em complemento de 2 com redução das linhas dos produtos parciais. Foi mostrado que as mesmas arquiteturas *array* sem sinal nos códigos Binário e Híbrido na base 2^m podem ser facilmente adaptadas para operações com sinal em complemento de 2. Como pôde ser mostrado, as nossas arquiteturas conseguem manter a mesma regularidade de um multiplicador *array* convencional que possibilita uma redução significativa de sinais de *glitching* e conseqüentemente o consumo de potência em relação ao algoritmo Booth Modificado;

- Implementação de algoritmos em linguagem de programação C. A implementação destes algoritmos teve por objetivo criar as condições de investigação dos métodos de correlação dos sinais, atividade de transição dos diferentes

códigos e criação de operadores regulares em formato BLIF. Com estas finalidades, os seguintes algoritmos foram implementados:

- algoritmo para a produção em formato PLA de cada um dos códigos estudados. Teve por objetivo a identificação da complexidade de cada um dos códigos, antes da implementação dos operadores aritméticos;
- algoritmo para o cálculo da atividade de transição dos códigos. Permitiu a identificação do potencial de redução da atividade de transição de cada um dos códigos estudados;
- algoritmo para a produção dos sinais de entrada dos operadores. A correlação dos dados está associada ao tipo de sinal de entrada dos operadores. Desta forma, foi implementado um algoritmo para produção de sinais de entrada tipo senos para estudo do efeito da correlação no consumo de potência dos operadores. Neste mesmo algoritmo foram também produzidos sinais pseudo-aleatórios para as entradas dos operadores aritméticos;
- algoritmo para a produção de operadores em formato BLIF. Nesta etapa do trabalho foram implementados circuitos somadores e multiplicadores em formato BLIF utilizado pelo ambiente SIS. O algoritmo implementado é genérico de tal forma que operadores somadores e multiplicadores Binários e Híbridos para qualquer largura de bits dos operandos e qualquer valor de m podem ser produzidos;
- algoritmo para a produção de arquiteturas FFT. Na área de processamento digital de sinais, algumas aplicações de sistemas de comunicação digital, tais como modems e terminais móveis de comunicação sem fio, necessitam de arquiteturas FFT com maior quantidade de pontos [HE 98, LI 99]. Desta forma, neste trabalho foi implementado um algoritmo para a produção de arquiteturas FFT seqüenciais no formato BLIF para diferentes números de amostras. A produção das arquiteturas neste formato permite a verificação de parâmetros de área e atraso, além da utilização de conversor para o formato da ferramenta SLS para verificação do consumo de potência das arquiteturas.

- Implementação de diferentes arquiteturas específicas para os algoritmos de filtro FIR e FFT. A implementação de diferentes arquiteturas de circuitos incluindo a sua exploração arquitetural, considerando aspectos de área, atraso e consumo de potência servem como parâmetro para a escolha da melhor alternativa arquitetural destes circuitos para determinadas aplicações. A metodologia consistiu na otimização de potência nos circuitos implementados a partir da sua exploração arquitetural com a utilização de técnicas de redução de potência (*pipelining, retiming*) associadas à utilização dos operadores aritméticos em código Híbrido. Como foi observado, principalmente a utilização da técnica *pipelining* nas arquiteturas implementadas, oferece uma significativa redução de potência, pela redução da atividade de *glitching* ao longo dos circuitos;

- Implementação de técnicas de baixa potência considerando a redução na atividade de transição dos circuitos. Na área de processamento digital de sinais há

um grande número de operações aritméticas envolvido. Desta forma, as diferentes técnicas de redução de potência estudados tiveram por objetivo a redução da atividade de chaveamento dos módulos aritméticos. Além disto, dada a natureza dos algoritmos da área de DSP que envolvem o produto de dados com apropriados coeficientes, procurou-se explorar o ordenamento ótimo dos coeficientes de forma a minimizar o consumo de potência dos circuitos. Neste trabalho foi implementado um algoritmo de ordenação e particionamento dos coeficientes baseado no algoritmo proposto em [MEH 95, MEH 96, MEH 98]. Neste algoritmo, os coeficientes são aplicados em arquiteturas de filtro FIR ordenados em uma arquitetura totalmente seqüencial. No nosso trabalho esta técnica foi utilizada em uma arquitetura de filtro FIR totalmente seqüencial e, adicionalmente, aplicamos esta técnica nas arquiteturas FFT seqüencial e semi-paralela com os coeficientes particionados. A técnica se mostrou particularmente mais eficiente no algoritmo FFT, onde se tem um maior grau de oportunidade para ordenação dos coeficientes em cada estágio da FFT. Como contribuição adicional, implementamos um algoritmo de ordenação e particionamento de coeficientes que foi aplicado nas arquiteturas de filtro FIR semi-paralelas. Este algoritmo se mostrou mais eficiente do que o proposto em [MEH 95, MEH 96, MEH 98], proporcionando uma maior redução de potência nas arquiteturas de filtro FIR semi-paralelas.

8.2 Trabalhos Futuros

Esta seção sugere algumas idéias de futuros trabalhos que darão prosseguimento à linha de pesquisa seguida nesta tese de doutorado.

8.2.1 Implementação dos Multiplicadores *Array* no Nível Físico

Como a dissipação de potência vem se tornando um dos parâmetros mais importante no projeto de circuitos CMOS, alguns fatores passam a ser fundamentais no projeto destes circuitos, tais como os limites teóricos e práticos que possam definir o desempenho de um circuito integrado, além de parâmetros físicos, tais como as interconexões entre os diferentes elementos do circuito, que têm se tornado um fator de dissipação de potência cada vez mais dominante.

Os operadores aritméticos *array* propostos neste trabalho apresentam o mesmo nível de regularidade de um multiplicador *array* convencional [HWA 79]. Esta característica pode tornar atrativa a implementação destas arquiteturas no nível físico (*layout*). Neste nível de abstração, pode-se ter a identificação mais precisa do percentual de redução de potência que pode ser obtido nos operadores de baixa potência propostos neste trabalho.

8.2.2 Exploração da Capacitância de Carga

Embora este trabalho tenha dado um indicativo da importância da aplicação dos operadores Híbridos para diferentes valores de carga nos barramentos, faz-se necessário uma maior exploração do verdadeiro potencial de redução de potência considerando este aspecto. Desta forma, uma das linhas de pesquisa a ser seguida como conseqüência dos resultados obtidos neste trabalho é a verificação do impacto

de redução de potência que pode ser obtido pelos operadores em código Híbrido a partir da aplicação de diferentes valores de capacitância de carga nos barramentos das arquiteturas de filtro FIR e FFT. Este estudo deverá ser realizado particularmente no baixo nível de abstração, após a implementação dos operadores no nível físico.

8.2.3 Aplicação de Circuitos Somadores Eficientes

Com o aspecto de regularidade das arquiteturas de multiplicadores mostradas neste trabalho, torna-se possível a aplicação de outras técnicas de redução de potência a estas arquiteturas. Desta forma, uma das técnicas sugeridas como trabalho futuro é a aplicação de circuitos somadores mais eficientes para redução do caminho crítico e das transições espúrias que são propagadas ao longo do *array*. Algumas técnicas como *carry-save adder arrays*, *Wallace trees* e *Dadda parallel counters* [CAP 83, KHA 96, MEI 99] já foram anteriormente utilizadas para tornar mais rápida a soma dos produtos parciais no multiplicador Booth. Desta forma, propõe-se como trabalho futuro a aplicação de algumas destas técnicas nas arquiteturas dos multiplicadores *array* para verificação do impacto nas reduções dos valores de atraso e consumo de potência que pode ser proporcionado. Outra abordagem, que já foi anteriormente proposta na literatura, refere-se à aplicação da lógica de transistores de passagem para aumento do desempenho nos circuitos somadores do multiplicador Booth [YAN 90, KHA 96, GOT 97]. Com a realização física dos nossos operadores aritméticos, esta mesma abordagem pode ser explorada.

8.2.4 Manipulação de Coeficientes

Neste trabalho foi apresentada uma técnica de manipulação de coeficientes para o melhor particionamento dos coeficientes das arquiteturas de filtros FIR, com a redução da distância de *Hamming* em grupos de coeficientes. O algoritmo proposto utiliza um método exaustivo de busca do melhor particionamento dos coeficientes. Embora tenha sido mostrado que esta técnica pode reduzir o consumo de potência das arquiteturas implementadas, a sua utilização em circuitos que utilizem uma maior quantidade de coeficientes se torna menos atrativa, devido ao longo tempo de computação envolvido. Desta forma, como trabalho futuro, sugere-se a utilização de um algoritmo baseado em heurísticas para aplicação em circuitos de filtros FIR e FFT com maiores quantidades de coeficientes.

Um outro enfoque que poderá ser explorado é a abordagem do impacto do consumo de potência a partir da redução da quantidade de coeficientes que pode ser utilizado, sem comprometer a resposta do filtro FIR. Neste caso, o objetivo é estabelecer um relacionamento entre o número de coeficientes necessários que mantenha a resposta do filtro dentro de uma determinada faixa de erro permitido na variação da amplitude do sinal na banda de passagem (*passband ripple*) e na faixa de atenuação do sinal (*stopband attenuation*). O aspecto principal é estabelecer uma estatística do consumo de potência de diferentes arquiteturas dos filtros com diferentes respostas dos sinais dentro da faixa de erro pré-estabelecida. A redução do comprimento da palavra nestes coeficientes é outro aspecto que pode ser explorado no sentido de redução da atividade de chaveamento dos circuitos implementados.

8.2.5 Aplicação dos Operadores Aritméticos *Array* em Arquiteturas mais Complexas

Neste trabalho foram implementadas arquiteturas de filtro FIR de 8^a ordem e arquiteturas FFT de 16 e 64 pontos. Como pode ser observado pelos resultados obtidos, a aplicação de operadores aritméticos em código Híbrido, particularmente com valor de m=2, apresentou valores de atraso e consumo de potência próximos às arquiteturas com operadores aritméticos com operadores Binários. Este aspecto está relacionado com a maior correlação dos dados de entrada dos multiplicadores em código Híbrido, além da menor atividade de transição dos coeficientes com este tipo de codificação. Desta forma, como trabalho futuro, pretende-se investigar arquiteturas de filtros FIR com maior número de *taps* e arquiteturas FFT com maior número de pontos para observação da correlação dos dados e influência dos coeficientes no consumo de potência com a utilização de operadores aritméticos Binário e Híbrido. Nestas arquiteturas mais complexas, outro enfoque que pode ser explorado é a ordenação e particionamento dos coeficientes, que como observado neste trabalho, apresenta um maior impacto nas arquiteturas com operadores aritméticos e coeficientes em código Híbrido. Em particular, as arquiteturas semi-paralelas terão um maior enfoque, pois como foi observado nas implementações mostradas, estas arquiteturas apresentam menores valores de consumo de potência.

Bibliografia

[AGH 2002] AGHAGHIRI, Y.; FALLAH, F.; PEDRAM, M. EZ Encoding: a class of irredundant low power codes for data address and multiplexed address buses. In: DESIGN, AUTOMATION AND TEST IN EUROPE, DATE, 2002, Paris. **Designers' forum**. [Los Alamitos: IEEE Computer Society], 2002. p.1102.

[ALI 94] ALIDINA, M. et al. Precomputation-Based Sequential Logic Optimization for Low Power. In: INTERNATIONAL WORKSHOP ON LOW POWER DESIGN, 1994, Napa Valley. **Proceedings...** New York: ACM, 1994. p.57–62.

[ALV 98] ALVANDPOUR, A.; EDEFORS, P.; SVENSSON, C. Separation and Extraction of Short-Circuit Power Consumption in Digital CMOS VLSI Circuits. In: IEEE INTERNATIONAL SYMPOSIUM ON LOW-POWER AND ELECTRONIC DESIGN AND SYSTEMS, ISLPED, 1998, Monterey. **Proceedings...** New York: ACM, 1998. p.245–249.

[ANG 96] ANGEL, E.; SWARTZLANDER, E. Low Power Parallel Multipliers. In:WORKSHOP ON VLSI SIGNAL PROCESSING, 9., 1996, San Francisco. **Proceedings...** Piscataway: IEEE Signal Processing Society, 1996. p.199–208.

[ASA 90] ASATO, C.; DITZEN, C.; DHOLAKIA, S. A Data-Path Multiplier with Automatic Insertion of Pipeline Stages. **IEEE Journal of Solid-State Circuit**, New York, v.25, p.383–387, 1990.

[ATH 94] ATHAS, W.; SVENSSON, J. Low-Power Digital Systems Based on Adiabatic-Switching Principles. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.3, n.4, p.398–407, Dec. 1994.

[AUS 98] AUSTRIA MIKRO SYSTEME INTERNATIONAL. 0.8μm CMOS Joint Group Design Rules. Austria, 1998.

[BAA 2000] BAAS, M. **An Approach to Low-Power, High-Performance, Fast Fourier Transform Processor Design**. Stanford: Stanford University, Department of Electrical Engineering, 2000. Relatório Técnico.

[BEN 99] BENINI, L.; DEMICHELI, G. System-Level Power Optimization: techniques and tools. In: INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONIC AND DESIGN, ISLPED, 1999, San Diego. **Proceedings...** New York: ACM, 1999. p.288–293.

[BEN 2001] BENINI, L.; DEMICHELI, G.; MACII, E. Designing Low-Power Circuits: Practical Recipes. **IEEE Circuits and Systems Magazine**, Notre Dame, v.1, n.1, p.7–25, Apr.2001.

[BEN 98] BENINI, L.; DEMICHELLI, G.; MACII, E. Power Optimization of Core-Based Systems by Address Bus Encoding. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.6, n.4, p.554–562, Dec. 1998.

[BEN 97] BENINI, L. et al. System-Level Power Optimization of Special Purpose Applications: the beach solution. In: INTERNATIONAL SYMPOSIUM ON LOW POWER AND ELECTRONIC DESIGN, ISLPED, 1997, Monterey. **Proceedings...** New York: ACM, 1997. p.24–29.

[BEN 96] BENINI, L.; SIEGEL, P.; DEMICHELI, G. Automatic Synthesis of Gated-Clocks Finite State Machines. **IEEE Transactions on Computer-Aided Design**, Stanford, v.15, n.6, p.630–646, June 1996.

[BHA 2000] BHAVNAGARWALA, A. et al. A Minimum Total Power Methodology for Projecting Limits on CMOS GSI. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.8, n.3, p.235–251, June 2000.

[BOO 51] BOOTH, A. A Signed Binary Multiplication Technique. **Journal of Mechanics and Applied Mathematics**, [S.l.], n.4, p.236–240, June 1951.

[BUR 95] BURR, J.; CHEN, Z.; BAAS, B. Ultra-Low Power CMOS Technology and Applications. In: **Low-Power HF Microelectronics**: unified approach. London: The Institution of Electrical Engineers, 1995.

[BUR 91] BURR, J.; PETERSON, A. Ultra low power CMOS technology. In: NASA SYMPOSIUM ON VLSI DESIGN, 1991. **Proceedings...** [S.l.: s.n.], 1991. p.4.2.1–4.2.13.

[BUR 94] BURR, J.; SHOTT, J. A 200mV Self-Testing Encoder/Decoder using Stanford Ultra-Low-Power CMOS. In: INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE, 1994, San Francisco. **Proceedings...** New York: Solid-State Circuit Society, 1994. p.84–85.

[CAB 2000] CABODI, G.; QUER, S.; SOMENZI, F. Optimizing Sequential Verification by Retiming Transformations. In: DESIGN AUTOMATION CONFERENCE, DAC, 37., 2000, Los Angeles. **Proceedings...** New York: ACM, 2000. p.601–606.

[CAL 93] CALLAWAY, T.; SWARTZLANDER, E. Optimizing multipliers for WSI. In: ANNUAL IEEE INTERNATIONAL CONFERENCE ON WAFER SCALE INTEGRATION, 5., 1993, San Francisco. **Proceedings...** New York: IEEE, 1993. p.85–94.

[CAP 83] CAPELLO, P.; STEIGLITZ, K. A VLSI Layout for a Pipelined Dadda Multiplier. **Transactions on Computers Systems**, New York, v.1, n.2, p.157–174, May 1983.

[CHA 95] CHANDRAKASAN, A.; BRODERSEN, R. **Low Power Digital CMOS Design**. Dordrecht: Kluwer Academic, 1995.

[CHA 92] CHANDRAKASAN, A. et al. HYPER-LP: a System for Power Minimization Using Architectural Transformations. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED-DESIGN, ICCAD, 1992, Santa Clara. **Proceedings...** New York: ACM, 1992. p.300–303.

[CHA 95a] CHANDRAKASAN, A. et al. Optimizing Power Using Transformations. **IEEE Transactions on Computer-Aided Design**, Stanford, v.14, n.1, p.12–31, Jan. 1995.

[CHA 92a] CHADRAKASAN, A.; SHENG, S.; BRODERSEN, R. Low-Power CMOS Digital Design. **IEEE Journal of Solid-State Circuits**, New York, v.27, n.4, p.473–484, Apr. 1992.

[CHA 2000] CHANG, N.; KIM, K.; CHO, J. Bus Encoding for Low-Power High-Performance Memory Systems. In: DESIGN AUTOMATION CONFERENCE, DAC, 37., 2000, Los Angeles. **Proceedings...** New York: ACM, 2000. p.800–805.

[CHA 99] CHANG, Y.; PARHI, K. Efficient FFT Implementation Using Digit-Serial Arithmetic. In: IEEE WORKSHOP ON SIGNAL PROCESSING SYSTEMS, 1999, Taipei. **Proceedings...** Piscataway: IEEE Signal Processing Society, 1999. p.645–653.

[CHA 93] CHATTERJEE, A.; ROY, R. An Architectural Transformation Program for Optimization of Digital Systems by Multi-Level Decomposition. In: DESIGN AUTOMATION CONFERENCE, DAC, 30., 1993, Dallas. **Proceedings...** New York: ACM, 1993. p.343–348.

[CHE 2000] CHENG, W.; PEDRAM, M. Power-Optimal Encoding for DRAM Address Bus. In: IEEE INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN, ISLPED, 2000, Rapallo. **Proceedings...** New York: ACM, 2000. p.250–252.

[CHE 96] CHERKAUER, B.; FRIEDMAN, E. A Hybrid Radix-4/Radix-8 Low Power, High Speed Multiplier Architecture for Wide Bit Widths. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1996. **Papers**. Piscataway: IEEE, 1996. v.4, p.53–56.

[CHU 94] CHUNG, J.; PARHI, K. Pipelining of Lattice IIR Digital Filters. **IEEE Transactions on Signal Processing**, Piscataway, v.42, n.4, p.751–761, Apr. 1994.

[CIR 87] CIRIT, M. Estimating Dynamic Power Consumption of CMOS Circuits. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1987, Bologna. **Proceedings...** New York: ACM, 1987. p.534–537.

[CON 97] CONG, J.; HE, L.; KOH, C. **Low Power design in deep submicron electronics - Layout Optimization**. Dordrecht: Kluwer Academic , 1997.

[COO 65] COOLEY, J.; TUKEY, J. An Algorithm for the Machine Calculation of Complex Fourier Series. In: MATHEMATICS OF COMPUTATION, Washington, v.19, p.297–301, Apr. 1965.

[COS 2000] COSTA, E.; BAMPI, S. **Técnicas para Análise e Redução de Potência a partir de Especificações Algorítmicas e Exploração Arquitetural**. 2000. 148f. Exame de Qualificação (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[COS 99] COSTA, E.; CARRO, L.; BAMPI, S. **Avaliação de Potência em Circuitos CMOS Digitais**. 1999. 105f. Trabalho Individual de Pesquisa (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[COS 2000a] COSTA, E. et al. Modeling of Short Circuit Power Consumption Using Timing-Only Logic Cell Macromodels. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 13., 2000, Manaus. **Proceedings...** Los Alamitos: IEEE Computer Society, 2000. p.222–227.

[COS 2001] COSTA, E.; MONTEIRO, J.; BAMPI, S. Power Efficient Arithmetic Operand Encoding. In:SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 14., 2001, Pirenópolis. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.201–206.

[COS 2001a] COSTA, E.; MONTEIRO, J.; BAMPI, S. Power Optimization Using Coding Methods on Arithmetic Operators. In: IEEE INTERNATIONAL SYMPOSIUM ON SIGNALS CIRCUITS AND SYSTEMS, SCS, 2001, Iasi. **Proceedings...** Piscataway: IEEE CAS Society, 2001. p.505–508

[COS 2002] COSTA, E.; MONTEIRO, J.; BAMPI, S. A New Architecture for 2's Complement Gray Encoded Array Multiplier . In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 15., Sept. 2002. **Proceedings...** Artigo aceito.

[COS 2002a] COSTA, E.; MONTEIRO, J.; BAMPI, S. A New Architecture for Signed Radix-2^m Pure Array Multipliers. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, Sept. 2002. **Proceedings...** Artigo aceito.

[COS 2002b] COSTA, E.; MONTEIRO, J.; BAMPI, S. A New Architecture for 2's Complement Pipelined Array Multiplier. In: IEEE INTERNATIONAL ASIC/SOC CONFERENCE, Sept. 2002. **Proceedings...** Artigo aceito.

[COS 2002c] COSTA, E.; MONTEIRO, J.; BAMPI, S. FIR Filter Design using Low Power Arithmetic Operators. In: IEEE DESIGN AND DIAGNOSTICS OF ELECTRONIC CIRCUITS AND SYSTEMS, DDECS, 2002, Brno. **Proceedings...** Los Alamitos: IEEE Computer Society, 2002. p.314–317.

[DEN 94] DENKER, J. A Review of Adiabatic Computing. In: IEEE SYMPOSIUM ON LOW POWER ELECTRONICS, 1994, San Diego. **Proceedings...** New York: ACM, 1994. p.10–12.

[DEV 95] DEVADAS, S.; MALIK, S. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. In: DESIGN AUTOMATION CONFERENCE, DAC, 32., 1995, San Francisco. **Proceedings...** New York: ACM, 1995. p.242–247.

[DEV 96] DEVADAS, S.; MALIK, S. Power Minimization in IC Design. **Transactions on Design Automation of Electronic System**, Washington, v.1, p.251–279, Jan. 1996.

[DIC 95] DICKINSON, A.; DENKER, S. Adiabatic Dynamic Logic. **IEEE Journal of Solid-State Circuits**, New York, v.30, n.3, p.311–315, Mar. 1995.

[DIN 98] DING, C.; YING, C.; PEDRAM, M. Gate-Level Power Estimation Using Tagged Probabilistic Simulation. **IEEE Transactions on Computer-Aided Design**, Stanford, v.17, n.11, p.1099–1107, Nov. 1998.

[ERD 96] ERDOGAN, A.; ARSLAN, T. **Low Power Multiplication Scheme for FIR Filter Implementation on Single Multiplier CMOS DSP Processors**. Cardiff, UK: School of Engineering, Wales University, 1996. (Electronic Letters On-line, n.19961298).

[ERD 2000] ERDOGAN, A.; ARSLAN, T. High Throughput FIR Filter Design for Low Power SOC Applications. In: ANNUAL INTERNATIONAL ASIC/SOC CONFERENCE, 13., 2000, Arlington. **Proceedings...** New York: IEEE, 2000. p.21–24.

[FAV 95] FAVALLI, M.; BENINI, L. Analysis of Glitch Power Dissipation in CMOS ICs. In: INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN, ISLPED, 1995, Dana Point. **Proceedings...** New York: ACM, 1995. p.123–128.

[GAL 94] GALLAGHER, W.; SWARTZLANDER, E. High Radix Booth Multipliers Using Reduced Area Adder Trees. In: ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS AND COMPUTERS, 28.,

1994, Monterey. **Proceedings...** Piscataway: IEEE Signal Processing Society, 1994. p.545–549.

[GEN 89] GENDEREN, A. SLS: an efficient switch-level timing simulator using min-max voltage waveforms. In: INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, VLSI, 1989, Munich. **Proceedings...** New York: ACM, 1989. p.79–88.

[GHO 92] GHOSH, A. et al. Estimation of Average Switching Activity in Combinational and Sequential Circuits. In: DESIGN AUTOMATION CONFERENCE, DAC, 29., 1992, Anaheim. **Proceedings...** New York: ACM, 1992. p.253–259.

[GOL 2000] GOLDOVSKY, A. et al. Design and Implementation of a 16 by 16 Low-Power Two's Complement Multiplier. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2000. **Papers**. Piscataway: IEEE, 2000. v.5, p.345–348.

[GON 97] GONZALES, R.; GORDON, M.; HOROWITZ, M. Supply and Threshold Voltage Scaling for Low Power CMOS. **IEEE Journal of Solid-State Circuits**, New York, v.32, n.8, p.1210–1216, Ago. 1997.

[GOO 94] GOODBY, L.; ORAILOGLU, A.; CHAU, P. Microarchitectural Synthesis of Performance-Constrained, Low Power VLSI Designs. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, 1994, Cambridge. **Proceedings...** Los Alamitos: IEEE Computer Society, 1994. p.323–330.

[GOR 95] GORMAN, S.; WILLS, J. Partial Column FFT Pipelines. **IEEE Transactions on Circuits and Systems II**, La Jolla, CA, v.42, n.6, p.414–423, June 1995.

[GOT 97] GOTO, G. et al. A 4.1-ns Compact 54 × 54-b Multiplier Utilizing Sign-Select Booth Encoders. **IEEE Journal of Solid-State Circuits**, New York, v.32, n.10, p.1676–1682, Oct. 1997.

[HAK 99] HAKENES, R.; MANOLI, Y. A segmented gray code for low-power microcontroller address buses. In: EUROMICRO CONFERENCE, 1999, Milan. **Proceedings...** Sankt Augustin: IEEE, 1999. v.1, p.240–243.

[HAM 99] HAMILTON, S. Taking Moore's Law Into the Next Century. **IEEE Transactions on Computers**, New York, v.32, p.43–73, Jan. 1999.

[HAR 89] HARTLEY, R.; CASAVANT, A. Tree-height minimization in pipelined architectures. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1989, Santa Clara. **Proceedings...** IEEE CAS Society: Piscataway, 1989. p.112–115.

[HAR 95] HARTLEY, R.; PARHI, K. **Digit-Serial Computation**. Dordrecht: Kluwer Academic, 1995.

[HAS 2002] HASAN, M.; ARSLAN, T. Scheme for reducing size of coefficient memory in FFT processor. **Electronic Letters**, United Kingdom, v.38, n.4, p.163–164, Feb. 2002.

[HAS 98] HASSOUN, S.; EBELING, C. Using Precomputation in Architecture and Logic Resynthesis. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1998, San Jose. **Proceedings...** Piscataway: IEEE CAS Society, 1998. p.316–323.

[HE 98] HE, S.; TORKELSON, M. Design and Implementation of a 1024-point Pipeline Processor. In: IEEE CUSTOM INTEGRATED CIRCUITS CONFERENCE, 1998, Santa Clara. **Proceedings...** New York: IEEE, 1998. p.131–134.

[HEN 2001] HENKEL, J.; LEKATSAS, H. A^2BC: Adaptive Address Bus Coding for Low Power Deep Sub-Micron Design. In: DESIGN AUTOMATION CONFERENCE, DAC, 38., 2001, Las Vegas. **Proceedings...** New York: ACM, 2001. p.744–749.

[HIR 96] HIRATA, A.; ONODERA, H.; TAMARU, K. Estimation of Short-Circuit Power Dissipation and Its Influence on Propagation Delay for Static CMOS Gates. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1996, Atlanta. **Papers**. Piscataway: IEEE, 1996. v.4, p.751–754.

[HIR 98] HIRATA, A.; ONODERA, H.; TAMARU, K. Estimation of Propagation Delay Considering Short-Circuit Current for Static CMOS Gates. **IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications**, Piscataway, v.45, n.11, p.1194–1198, Nov. 1998.

[HUA 99] HUANG, J.; LENG, T. Generalized loop-unrolling:a method for program speedup. In: SYMPOSIUM ON APPLICATION-SPECIFIC SYSTEMS AND SOFTWARE ENGINEERING AND TECHNOLOGY, ASSET, 1999, Richardson, Texas. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.244–248.

[HUA 94] HUANG, S.; RABAEY, J. Maximizing the Throughput of High Performance DSP Applications Using Behavioral Transformations. In: EUROPEAN DESIGN AUTOMATION CONFERENCE, EURO-DAC, 1994, Grenoble. **Proceedings...** New York: ACM, 1994. p.25–30.

[HWA 79] HWANG, K. **Computer Arithmetic**: Principles, Architecture and Design. New York: John Wiley and Sons, 1979.

[ITR 99] ITRS. **The International Technology Roadmap for Semiconductors**. Disponível em: *<http://public.itrs.net/files/1999.SIA.Roadmap/Home.htm>*. Acesso em: 04 jul.2001.

[JEF 2001] JEFREY, A. et al. Interconnect Limits on Gigascale Integration (GSI) in the 21st Century. Proceedings of the IEEE, New York, v.89, p.305–324, 2001.

[KHA 96] KHATER, I.; BELLAOUAR, A.; ELMASRY, M. Circuit Techniques for CMOS Low-Power High-Performance Multipliers. **IEEE Journal of Solid-State Circuits**, New York, v.31, n.10, p.1535–1546, Oct. 1996.

[KIM 2002] KIM, C.; ROY, K. Dynamic V_{TH} Scaling Scheme for Active Leakage Power Reduction. In: DESIGN, AUTOMATION AND TEST IN EUROPE DATE, 2002, Paris. **Proceedings...** Los Alamitos: IEEE Computer Society, 2002. p.163–167.

[KOM 98] KOMATSU, S.; IKEDA, M.; ASADA, K. Low Power Microprocessors for Comparative Study on Bus Architecture and Multiplexer Architecture. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 1998, Monterey. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998. p.323–324.

[KOS 97] KOSEKI, A.; KOMASTU, H.; FUKAZAWA, Y. A method for estimating optimal unrolling times for nested loops. In: INTERNATIONAL SYMPOSIUM ON PARALLEL ARCHITECTURES, ALGORITHMS, AND NETWORKS, ISPAN, 3., 1997, Taipei. **Proceedings...** Los Alamitos: IEEE Computer Society, 1997. p.376–382.

[KUM 95] KUMAR, N. et al. Profile-Driven Behavioral Synthesis for Low-Power VLSI Systems. **IEEE Design and Test of Computers**, California, v.12, n.3, p.70–84, Fall 1995.

[LAL 95] LALGUDI, K.; PAPAEFTHYMIOU, M. DelaY: an efficient tool for retiming with realistic delay modeling. In: DESIGN AUTOMATION CONFERENCE, DAC, 32., 1995, San Francisco. **Proceedings...** New York: ACM, 1995. p.304–309.

[LAN 94] LANDMAN, P. **Low-Power Architectural Design Methodologies**. 1994. [Tese]. University of California, Berkeley.

[LAN 98] LANDMAN, P. **A Survey of High-Level Power Estimation Techniques**. In: CHANDRAKASAN, A.; BRODERSEN, R. **Low Power CMOS Design Book**. Dordrecht: Kluwer Academic, 1998.

[LAN 95] LANDMAN, P.; RABAEY, J. Architectural Power Analysis: the Dual Bit Type Method. **IEEE Transactions on Very large Scale Integration (VLSI) Systems**, New York, v.3, n.2, p.173–187, June 1995.

[LAN 97] LANDMAN, P.; RABAEY, J. Activity-Sensitive Architectural Power Analysis. **IEEE Transactions on Computer-Aided Design**, Stanford, v.15, n.6, p.516–532, June 1997.

[LEE 97] LEE, M. et al. Power Analysis and Minimization Techniques for Embedded DSP Software. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.5, n.1, p.123–135, Mar. 1997.

[LEI 95] LEIJTEN, J.; MEERBERGEN, J. van; JESS, J. Analysis and reduction of glitches in synchronous networks. In: EUROPEAN DESIGN AND TEST CONFERENCE, 1995, Paris. **Proceedings...** New York: ACM, 1995. p.398–403.

[LEI 83] LEISERSON, C.; SAXE, J. Optimizing Synchronous Sistems. **IEE Journal of VLSI and Computer Systems**, [S.l.], v.1, n.1, p.41–67, 1983.

[LEM 94] LEMMONDS, C.; SHETTI, S. A low power 16 by 16 multiplier using transition reduction circuitry. In: INTERNATIONAL WORSHOP ON LOW POWER DESIGN, 1994, Napa Valley. **Proceedings...** New York: ACM, 1994. p.139–142.

[LI 99] LI, W.; WANHAMMAR, L. A Pipeline FFT Processor. In: IEEE WORKSHOP ON SIGNAL PROCESSING SYSTEMS, SIPS, 1999, Leicester. **Proceedings...** Piscataway: IEEE Signal Processing Society, 1999. p.654–662.

[MA 99] MA, Y. An Effective Memory Addressing Scheme for FFT Processors. **IEEE Transactions on Signal Processing**, Piscataway, v.47, n.3, p.907–911, Mar. 1999.

[MAR 94] MARTIN, R.; KNIGHT, J. **A Tutorial on Behavioral Synthesis Power Optimization**. Ottawa, Canadá: Carleton University, 1994.

[MAR 95] MARTIN, R.; KNIGHT, J. Power-Profiler: optimizing asics power consumption at the behavioral level. In: DESIGN AUTOMATION CONFERENCE, DAC, 32., 1995, San Francisco. **Proceedings...** New York: ACM, 1995. p.42–47.

[MEH 95] MEHENDALE, M.; SHERLEKAR, S.; VENKATESH, G. Techniques for Low Power Realization of FIR Filters. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 1995. **Proceedings...** New York: ACM, 1995. p.447–450.

[MEH 96] MEHENDALE, M.; SHERLEKAR, S.; VENKATESH, G. Low Power Realization of FIR Filters Using Multirate Architectures. In: INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, VLSI, 9., 1996, Bangalore. **Proceedings...** New York: ACM, 1996. p.370–375.

[MEH 98] MEHENDALE, M.; SHERLEKAR, S.; VENKATESH, G. Low Power Realization of FIR Filters on Programmable DSP's. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.6, n.4, p.546–553, Dec. 1998.

[MEH 98a] MEHENDALE, M.; SHERLEKAR, S.; VENKATESH, G. Algorithmic and Architectural Transformations for Low Power Realization of FIR Filters. In: INTERNATIONAL CONFERENCE ON ON VERY LARGE SCALE INTEGRATION, VLSI, 11., 1998, Chennai. **Proceedings...** New York ACM, 1998. p.12–17.

[MEH 96a] MEHRA, R. et al. **Algorithm and Architectural Level Methodologies for Low Power - Low Power Design Methodologies**. Dordrecht: Kluwer Academic, 1996.

[MEH 94] MEHRA, R.; RABAEY, J. Behavioral Level Power Estimation and Exploration . In: INTERNATIONAL WORKSHOP ON LOW POWER DESIGN, 1994, Napa Valley. **Proceedings...** New York: ACM, 1994. p.197–202.

[MEH 96b] MEHTA, H.; OWENS, R.; IRWIN, M. Some Issues in Gray Code Addressing. In: GREAT LAKES SYMPOSIUM ON VERY LARGE SCALE INTEGRATION, 1996, Ames. **Proceedings...** New York: ACM, 1996. p.178–181.

[MEI 99] MEIER, P.; RUTENBAR, R.; CARLEY, L. Inverse Polarity Techniques for High-Speed/Low-Power Multipliers. In: IEEE INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN, ISLPED, 1999. **Proceedings...** New York: ACM, 1999. p.264–266.

[MET 96] META SOFTWARE INC. **Hspice**: user's manual. Campbell, CA, 1996.

[MIL 92] MILLAR, B.; MADRID, P.; SWARTZLANDER, E. A Fast Hybrid Multiplier Combining Booth and Wallace/DADDA Algorithms. In: MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, 35., 1992, Washington. **Proceedings...** Piscataway: IEEE CAS Society, 1992. v.1, p.158–165.

[MOL 98] MOLONEY, D. et al. Low-Power 200-Msps, Area-Efficient, Five-Tap Programmable FIR Filter. **IEEE Journal of Solid-State Circuits**, New York, v.33, n.7, p.1134–1138, July 1998.

[MON 96] MONTEIRO, J. **A Computer-Aided Design Methodology for Low Power Sequential Logic Circuits**. 1996. [Tese]. Massachusetts Institute of Technology, Massachussets.

[MON 99] MONTEIRO, J. Power Optimization using Dynamic Power Management. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 12., 1999, Natal. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.134–139.

[MON 96a] MONTEIRO, J.; DEVADAS, S. **Computer-Aided Design Techniques for Low Power Sequential Logic Circuits**. Dordrecht: Kluwer Academic, 1996.

[MON 96b] MONTEIRO, J.; DEVADAS, S. Techniques for Power Estimation and Optimization at the Logic Level: a survey. **Journal of VLSI Signal Processing Systems**, Princeton, v.13, n.2/3, p.259–276, Aug./Sept. 1996.

[MON 93] MONTEIRO, J.; DEVADAS, S.; GHOSH, A. Retiming Sequential Circuits for Low Power. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1993, Santa Clara. **Proceedings...** Piscataway: IEEE CAS Society, 1993. p.398–402.

[MON 95] MONTEIRO, J. et al. Optimization of Combinational and Sequential Logic Circuits for Low Power Using Precomputation. In: CHAPEL HILL CONFERENCE ON ADVANCED RESEARCH ON VLSI, 1995, Chapel Hill. **Proceedings...** New York: IEEE, 1995. p.430–444.

[MON 96c] MONTEIRO, J. et al. Scheduling Techniques to Enable Power Management. In: DESIGN AUTOMATION CONFERENCE, DAC, 33., 1996, Las Vegas. **Proceedings...** New York: ACM, 1996. p.349–352.

[MON 98] MONTEIRO, J.; OLIVEIRA, A. Finite State Machine Decomposition for Low Power. In: DESIGN AUTOMATION CONFERENCE, DAC, 35., 1998, San Francisco. **Proceedings...** New York: ACM, 1998. p.758–763.

[MON 2000] MONTEIRO, J.; OLIVEIRA, A. FSM Decomposition by Direct Circuit Manipulation Applied to Low Power Design. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 2000. **Proceedings...** Los Alamitos: IEEE Computer Society, 2000. p.351–358.

[MOS 95] MOSHNYAGA, V.; TAMARU, K. A Comparative Study of Switching Activity Reduction Techniques for Design of Low-Power Multipliers. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1995. **Papers**. Piscataway: IEEE, 1995. v.3, p.1560–1563.

[MUH 2000] MUHAMMAD, K.; STASZEWSKI, R.; BALSARA, P. Low Power Techniques and Design Tradeoffs in Adaptive FIR Filtering for PRML Read Channels. In: INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN, ISLPED, 2000. **Proceedings...** New York: ACM, 2000. p.262–267.

[MUR 98] MURGAI, R.; FUJITA, M.; OLIVEIRA, M. Using Complementation and Resequencing to Minimize Transitions. In: DESIGN AUTOMATION CONFERENCE, DAC, 35., 1998, San Francisco. **Proceedings...** New York: ACM, 1998. p.694–697.

[MUS 95] MUSSOL, E.; CORTADELLA, J. High-Level Synthesis Techniques for Reducing the Activity of Functional Units. In: IN-

TERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONIC AND DESIGN, ISLPED, 1995, Dana Point. **Proceedings...** New York: ACM, 1995. p.99–104.

[NAC 2000] NACHTERGAELE, L.; TIWARI, V.; DUTT, N. System and Architecture-Level Power Reduction for Microprocessor-Based Communication and Multi-Media Applications. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2000, San Jose. **Proceedings...** Piscataway: IEEE CAS Society, 2000. p.569–573.

[NAJ 93] NAJM, F. Transition density: a new measure of activity in digital circuits. **IEEE Transactions on Computer-Aided Design**, Stanford, v.12, n.2, p.310–323, Feb. 1993.

[NAJ 90] NAJM, F. et al. Probabilistic Simulation for Reliability Analysis of CMOS Circuits. **IEEE Transactions on Computer-Aided Design**, Stanford, v.9, p.439–450, Apr. 1990.

[NAK 86] NAKAMURA, S. Algorithm for Iterative Array Multiplication. **IEEE Transactions on Computers**, Washington, v.35, n.8, p.713–719, Aug. 1986.

[NGU 2000] NGUYEN, H.; CHATTERJEE, A. Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis. **IEEE Transactions on Very Large Scale Integration (VLSI) System**, New York, v.8, n.4, p.419–424, Aug. 2000.

[NTR 97] NTRS. **The National Technology Roadmap for Semiconductors** Disponível em: *<http://jbsim.cs.pku.edu.cn/users/chengxu/Org_web_ext/PDF_FILES/1997Roadmap_all.pdf>*. Acesso em 04 jul. 2001.

[OPP 89] OPPENHEIM, A.; SCHAFER, R. **Discrete-Time Signal Processing**. London: Prentice Hall Signal International, 1989.

[PAR 95] PARHI, K. High-Level Algorithm and Architecture Transformations for DSP Synthesis. **Journal of VLSI Signal Processing Systems**, Princeton, v.9, n.1–2, p.121–143, Jan. 1995.

[PAR 2001] PARK, I.; KANG, H. Digital Filter Synthesis Based on Minimal Signed Digit Representation. In: DESIGN AUTOMATION CONFERENCE, DAC, 38., 2001, Las Vegas. **Proceedings...** New York: ACM, 2001. p.468–473.

[PAS 99] PASKO, R. et al. A New Algorithm for Elimination of Common Subexpressions. **IEEE Transactions on Computer-Aided Design**, Stanford, v.18, p.58–68, Jan. 1999.

[PEK 99] PEKMESTZI, K. Multiplexer-Based Array Multipliers. **IEEE Transactions on Computers**, New York, v.48, n.1, p.15–23, Jan. 1999.

[POP 2000] POPPEN, F. **Low Power Design Guide**. Oldenburg: Oldenburger Forschungs-Und Entwicklungsinstitut Fur Informatik-Werkzeuge Und-Systeme-OFFIS, 2000.

[POR 2000] PORTELA, J. **Editor Gráfico para Sistemas Computacionais**. Lisboa: Instituto Superior Técnico - Departamento de Engenahria Informática, 2000. Trabalho Final de Curso.

[POT 92] POTKONJAK, M. Pipelining: just another transformation. In: INTERNATIONAL CONFERENCE ON APPLICATION SPECIFIC ARRAY PROCESSORS, ASAP, 1992, Oakland. **Proceedings...** Los Alamitos: IEEE Computer Society, 1992. p.163–175.

[POT 94] POTKONJAK, M.; RABAEY, J. Optimizing Resources Utilization Using Transformations. **IEEE Transactions on Computer-Aided Design**, Stanford, v.13, n.3, p.277–292, Mar. 1994.

[POT 96] POTKONJAK, M.; SRIVASTAVA, M.; CHANDRAKASAN, A. Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpressions Elimination. **IEEE Transactions on Computer-Aided Design**, Stanford, v.15, p.151–165, Feb. 1996.

[RAB 96] RABAEY, J.; PEDRAM, M. **Low Power Design Methodologies**. Dordrecht: Kluwer Academic, 1996.

[RAG 99] RAGHUNATHAN, A.; DEY, S.; JHA, N. Register Transfer Level Power Optimization with Emphasis on Glitch Analysis and Reduction. **IEEE Transactions on Computer-Aided Design**, Stanford, v.18, p.1114–1131, Aug. 1999.

[RAG 94] RAGHUNATHAN, A.; JHA, N. Behavioral Synthesis for Low Power. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, 1994, Cambridge. **Proceedings...** Los Alamitos: IEEE Computer Society, 1994. p.318–322.

[RAG 95] RAGHUNATHAN, A.; JHA, N. An Iterative Improvement Algorithm for Low Power Data Path Synthesis. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1995, San Jose. **Proceedings...** New York: ACM, 1995. p.597–602.

[RAG 95a] RAGHUNATHAN, J.; PEDRAM, M. Register Allocation and Binding for Low Power. In: DESIGN AUTOMATION CONFERENCE, DAC, 32., 1995, San Francisco. **Proceedings...** New York: ACM, 1995. p.29–35.

[RAM 99] RAMOS, P.; OLIVEIRA, A. Low Overhead Encodings for Reduced Activity in Data and Address Buses. In: INTERNATIONAL SYMPOSIUM ON SIGNALS, CIRCUITS AND SYSTEMS, 1999, Iasi. **Proceedings...** Piscataway: IEEE CAS Society, 1999. p.21–24.

[RAM 97] RAMPRASAD, S.; SHANBHAG, N.; HAJJ, I. Analytical Estimation of Signal Transition Activity from Word-Level Statistics. **IEEE Transactions on Computer-Aided Design**, Stanford, v.16, n.7, p.718–733, July 1997.

[REI 60] REITWIESNER, G. Binary Arithmetic. **Advances in Computers**, New York, v.1, p.261–265, 1960.

[ROY 96] ROY, K.; JOHNSON, M. **Low Power Design in Deep Submicron Electronics - Software Design for Low Power**. Dordrecht: Kluwer Academic, 1996.

[SAC 98] SACHA, J.; IRWIN, M. Number Representations for Reducing Data Bus Power Dissipation. In: ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS AND COMPUTERS, 32., 1998, Pacific Grove. **Proceedings...** Piscataway: IEEE Signal Processing Society, 1998. p.213–217.

[SAM 90] SAM, H.; GUPTA, A. A Generalized Multibit Recoding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementations. **IEEE Transactions on Computers**, New York, v.39, n.8, p.1006–1015, July 1990.

[SAN 97] SANKARAYYA, N.; ROY, K.; BHATTACHARYA, D. Algorithm for Low Power and High Speed FIR Filter Realization Using Differential Coefficients. **IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing**, Piscataway, v.44, n.6, p.488–497, June 1997.

[SEI 2001] SEIDEL, P.; MCFEARIN, L.; MATULA, D. Binary Multiplication Radix-32 and Radix-256. In: SYMPOSIUM ON COMPUTER ARITHMETIC, 15., 2001. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.23–32.

[SEN 92] SENTOVICH, E. et al. **SIS**: a System for Sequential Circuit Synthesis. Berkeley: University of California, 1992.

[SHA 2002] SHAMS, A.; DARWISH, T.; BAYOUMI, M. Performance Analysis of Low-Power 1-Bit CMOS Full Adder Cells. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.10, n.1, p.20–29, Feb. 2002.

[SOL 2002] SOLD - Synopsys On Line Documentation. **Synopsys - User's Manual**. Disponível em: <*http://www.synopsys.com/home.html*>. Acesso em 04 jul. 2002.

[STA 95] STAN, M.; BURLESON, W. Bus-Invert Coding for Low-Power I/O. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.3, n.1, p.49–58, Mar. 1995.

[STA 97] STAN, M.; BURLESON, W. Low-Power Encodings for Global Communication in CMOS VLSI. **IEEE Transactions on Very Large Scale Integrated (VLSI) Systems**, New York, v.5, n.3, p.444–455, Mar. 1997.

[STA 97a] STAN, M.; BURLESON, W. Limited-Weight Codes for Low-Power I/O. In: INTERNATIONAL WORKSHOP ON LOW POWER DESIGN, 1997. **Proceedings...** New York: ACM, 1997. p.70–73.

[SU 95] SU, C.; DESPAIN, A. A Cache Design Tradeoffs for Power and Performance Optimization AQ Case Study. In: IEEE SYMPOSIUM ON LOW POWER DESIGN, 1995, Laguna. **Proceedings...** New York: ACM, 1995. p.63–68.

[SYL 2001] SYLVESTER, D.; KAUL, H. Future Performance Challenges in Nanometer Design. In: DESIGN AUTOMATION CONFERENCE, DAC, 38., 2001, Las Vegas. **Proceedings...** New York: ACM, 2001. p.3–8.

[SZW 94] SZWARC, V. et al. A Chip Set for Pipeline and Parallel Pipeline FFT Architectures. **Journal of VLSI Signal Processing Systems**, Princeton, v.8, p.253–265, Dec. 1994.

[TAE 2000] TAEKYOON, A. et al. A new cost model for high-level power optimization and its application. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2000, Geneva. **Papers**. Piscataway: IEEE, 2000. p.573–576.

[TAK 97] TAKI, K. et al. Super Low Power 8-bit CPU with Pass-Transistor Logic. In: PROCEEEDINGS OF THE ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 1997. **Proceedings...** Los Alamitos: IEEE Computer Society, 1997. p.663–664.

[TIW 96] TIWARI, V.; MALIK, S.; WOLFE, A. Instruction Level Power Analysis and Optimization of Software. **Journal of VLSI Signal Processing Systems**, Princeton, v.13, n.2, p.1–18, Aug. 1996.

[TSU 95] TSUI, C. et al. Power Estimation Methods for Sequential Logic Circuits. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.3, n.3, p.404–416, Sept. 1995.

[TSU 93] TSUI, C.; PEDRAM, M.; DESPAIN, A. Efficient Estimation of Dynamic Power Consumption under a Real Delay Model. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1993, Santa Clara. **Proceedings...** Piscataway: IEEE CAS Society, 1993. p.224–228.

[TUR 98] TURGIS, S.; AUVERGNE, D. A Novel Macromodel for Power Estimation in CMOS Structures. **IEEE Transactions on Computer-Aided Design**, Stanford, v.17, n.11, p.1090–1098, Nov. 1998.

[VEE 84] VEENDRICK, H. Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits. **IEEE Journal of Solid-State Circuits**, New York, v.19, p.468–473, Aug. 1984.

[VEE 95] VEERAMACHANESI, V.; TYAGI, A.; RAJGOPAL, S. Reencoding for Low Power State Assignment of FSM's. In: IEEE INTERNATIONAL SYMPOSIUM ON LOW POWER DESIGN, ISLPED, 1995, Dana Point. **Proceedings...** New York: ACM, 1995.

[VEM 94] VEMURU, S.; SCHEINBERG, N. Short-Circuit Power Dissipation Estimation for CMOS Logic Gates. **IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications**, Piscataway, v.41, n.11, p.762–765, Nov. 1994.

[WAL 64] WALLACE, C. A Suggestion for a Fast Multiplier. **IEEE Transactions on Electronic Computers**, [S.l.], v.13, p.14–17, Feb. 1964.

[WAN 95] WANG, C.; PARHI, K. High-Level DSP Synthesis Using Concurrent Transformations, Scheduling, and Allocation. **IEEE Transactions on Computer-Aided Design**, Stanford, v.14, n.3, p.274–295, Mar. 1995.

[WAN 2001] WANG, Y.; JIANG, Y.; SHA, E. On Area-Efficient Low Power Array Multipliers. In: IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS, ICECS, 8., 2001, Malta. **Proceedings...** Piscataway: IEEE CAS Society, 2001. p.1429–1432.

[WES 94] WESTE, N.; ESHRAGHIAN, K. **Principles of CMOS VLSI Design**. 2nd.ed. Boston: Addison-Wesley, 1994.

[WU 98] WU, A.; LIU, K. Algorithm-Based Low-Power Transform Coding Architectures: the multirate approach. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.6, n.4, p.707–718, Dec. 1998.

[WU 98a] WU, A.; NG, C.; TANG, K. Modified Booth pipelined multiplication. **Electronic Letters**, United Kingdom, v.34, n.2, p.1179–1180, June 1998.

[YAN 90] YANO, K.; AL. et. A 3.8ns CMOS 16x16 Multiplier Using Complementary Pass Transistor Logic. **IEEE Journal of Solid-State Circuits**, New York, v.25, n.2, p.388–395, Apr. 1990.

[YU 2000] YU, S.; SWARTZLANDER, E. A New Pipelined Implementation of the Fast Fourier Transform. In: ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS AND COMPUTERS, 34., 2000, Pacific Grove. **Proceedings...** Piscataway: IEEE Signal Processing Society, 2000. v.1, p.423–427.

[YU 2000a] YU, Z.; WASSERMAN, L.; WILLSON, A. A Painless Way to Reduce Power by Over 18% in Booth-Encoded Carry-Save Array Multipliers for DSP. In: WORKSHOP ON SIGNAL PROCESSING SYSTEMS, SIPS, 2000, Lafayette. **Proceedings...** Piscataway: IEEE Signal Processing Society, 2000. p.571–580.